The Pre-Shared Key Modes of HPKE

Joël Alwen\textsuperscript{1}, Jonas Janneck\textsuperscript{2}, Eike Kiltz\textsuperscript{2}, and Benjamin Lipp\textsuperscript{3}

\textsuperscript{1} AWS-Wickr, Seattle, USA
\texttt{alwenjo@amazon.com}
\textsuperscript{2} Ruhr-Universität Bochum, Germany
\texttt{jonas.janneck,eike.kiltz}@rub.de
\textsuperscript{3} Max Planck Institute for Security and Privacy, Bochum, Germany
\texttt{benjamin.lipp@mpi-sp.org}

October 17, 2023

Abstract. The Hybrid Public Key Encryption (HPKE) standard was recently published as RFC 9180 by the Crypto Forum Research Group (CFRG) of the Internet Research Task Force (IRTF). The RFC specifies an efficient public key encryption scheme, combining asymmetric and symmetric cryptographic building blocks.

Out of HPKE’s four modes, two have already been formally analyzed by Alwen et al. (EUROCRYPT 2021). This work considers the remaining two modes: HPKE\textsubscript{PSK} and HPKE\textsubscript{AuthPSK}. Both of them are “pre-shared key” modes that assume the sender and receiver hold a symmetric pre-shared key. We capture the schemes with two new primitives which we call pre-shared key public-key encryption (pskPKE) and pre-shared key authenticated public-key encryption (pskAPKE). We provide formal security models for pskPKE and pskAPKE and prove (via general composition theorems) that the two modes HPKE\textsubscript{PSK} and HPKE\textsubscript{AuthPSK} offer active security (in the sense of insider privacy and outsider authenticity) under the Gap Diffie-Hellman assumption.

We furthermore explore possible post-quantum secure instantiations of the HPKE standard and propose new solutions based on lattices and isogenies. Moreover, we show how HPKE’s basic HPKE\textsubscript{PSK} and HPKE\textsubscript{AuthPSK} modes can be used black-box in a simple way to build actively secure post-quantum/classic-hybrid (authenticated) encryption schemes. Our hybrid constructions provide a cheap and easy path towards a practical post-quantum secure drop-in replacement for the basic HPKE modes HPKE\textsubscript{Base} and HPKE\textsubscript{Auth}.

Keywords. Authenticated Public Key Encryption, Post-Quantum Hybrid, Open Standards, HPKE
# Table of Contents

1 Introduction ........................................................... 3  
   1.1 Our Contributions ................................................. 4  
2 Preliminaries .......................................................... 7  
   2.1 Notations .................................................................. 7  
   2.2 (Authenticated) Key Encapsulation Mechanisms .............. 8  
   2.3 Authenticated Public Key Encryption ......................... 9  
   2.4 Pseudorandom Functions ......................................... 11  
   2.5 Authenticated Encryption with Associated Data ............ 12  
   2.6 Digital Signatures ............................................... 13  
   2.7 Non-Interactive Key Exchange ................................. 14  
3 Pre-Shared Key (Authenticated) Encryption ...................... 14  
   3.1 Syntax .................................................................. 15  
   3.2 Privacy .............................................................. 18  
   3.3 Authenticity ......................................................... 19  
4 HPKE’s constructions of a pskPKE and pskAPKE ............... 19  
   4.1 Generic Constructions ............................................. 19  
   4.2 Security of pskPKE and pskAPKE ............................ 20  
   4.3 The Security of HPKE’s PSK Modes ........................... 22  
   4.4 Proof of Theorem 4 ................................................. 22  
5 Hybrid Post-Quantum APKE ......................................... 26  
6 Post-Quantum AKEM Constructions ................................ 28  
   6.1 KEM-then-Sign-then-Hash ....................................... 28  
   6.2 AKEM from NIKE ................................................ 29  
A Omitted Security Definition ......................................... 32  
   A.1 Active Security NIKE .............................................. 32  
B Proofs for the pskPKE and pskAPKE constructions ........... 32  
   B.1 Proof of Theorem 1 ................................................ 32  
   B.2 Proof of Theorem 2 ................................................ 38  
   B.3 Proof of Theorem 3 ................................................ 42  
C Proofs for the Hybrid Post-Quantum APKE ....................... 44  
   C.1 Proof of Theorem 5 ................................................ 44  
   C.2 Proof of Theorem 6 ................................................ 48  
D Proof for the AKEM Constructions .................................. 52  
   D.1 Proof of Theorem 7 ................................................ 52  
   D.2 Proof of Theorem 8 ................................................ 54  
   D.3 Proof of Theorem 9 ................................................ 55  
   D.4 Proof of Theorem 10 ............................................... 56
1 Introduction

The Hybrid Public Key Encryption (HPKE) standard was published as RFC 9180 [4] by the Crypto Forum Research Group (CFRG) of the Internet Research Task Force (IRTF) in February 2022. The RFC specifies an efficient public key encryption scheme, combining asymmetric and symmetric cryptographic building blocks. While this an old and relatively well understood paradigm, the new standard was developed in an effort to address issues in previous standardizations of hybrid public key encryption. For example, HPKE relies on modern cryptographic building blocks, provides test vectors to ease development of interoperable implementations, and already received some cryptographic analysis during its development, inspired by the “analysis-prior-to-deployment” design philosophy adopted for the development of the TLS 1.3 protocol [20]. At the time of development of HPKE, two IETF standardization efforts already started building upon it: the Messaging Layer Security (MLS) protocol [3], and the Encrypted Client Hello privacy extension of TLS 1.3 [21]. Since its publication, HPKE has also been adopted by other higher-level protocols, like the published RFC 9230 Oblivious DNS over HTTPS [16], and the Distributed Aggregation Protocol for Privacy Preserving Measurement [15], and thus, has become an important building block of today’s and the future Internet.

The HPKE standard may appear to resemble a “public key encryption” approach, aligning with the KEM/DEM paradigm [9]. Indeed, it incorporates a Key Encapsulation Mechanism (KEM) and an Authenticated Encryption with Associated Data (AEAD), functioning as a Data Encapsulation Mechanism (DEM) based on the KEM/DEM paradigm. However, upon closer inspection HPKE turns out to be more complex than this perfunctory description implies. First, HPKE actually consists of 2 different KEM/DEM constructions. Moreover, each construction can also be instantiated with a pre-shared key (psk) known to both sender and receiver, which is used in the key schedule KS to derive the DEM key. In total this gives rise to 4 different modes for HPKE.

The basic mode HPKEBase makes use of a standard KEM to obtain a “message privacy and integrity” only mode. This mode can be extended to HPKEPSK to support authentication of the sender via a psk. The remaining 2 HPKE modes make use of a different KEM/DEM construction built from a rather non-standard KEM variant called Authenticated KEM (AKEM) [1]. An AKEM can be thought of the KEM analogue of signcryption [22]. In particular, sender and receiver both have their own public/private keys. Each party requires their own private and the other party’s public key to perform en/decryption. The AKEM-based HPKE modes also intend to authenticate the sender to the receiver. Just as in the KEM-based case, the AKEM/DEM construction can be instantiated in modes either without psk (HPKEAuth) or with a psk (HPKEAuthPSK).

The HPKE RFC constructs a KEM and an AKEM based on specific Diffie-Hellman groups (such as P-256, P-384, P-521 NIST curves [19], Curve25519, or Curve448 [17]). Alwen, Blanchet, Hauck, Kiltz, Lipp, and Riepel [1] have analyzed the security of the Diffie-Hellman AKEM and showed that it can be securely combined with a key schedule KS and an AEAD to obtain concrete security bounds for the HPKEAuth modes, as defined in the HPKE standard, see Table 1. Their work explicitly leaves analyzing the remaining two HPKE modes HPKEPSK and HPKEAuthPSK for future work.

4 https://irtf.org/cfrg
Table 1: HPKE modes and their security.

<table>
<thead>
<tr>
<th>HPKE mode</th>
<th>Authenticated?</th>
<th>PSK?</th>
<th>Primitive</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPKE Base</td>
<td>–</td>
<td>–</td>
<td>PKE</td>
<td>CCA_PKE (folklore)</td>
</tr>
<tr>
<td>HPKE PSK</td>
<td>–</td>
<td>√</td>
<td>pskPKE ($\S3$)</td>
<td>CCA &amp; Auth ($\S4$)</td>
</tr>
<tr>
<td>HPKE AuthPSK</td>
<td>√</td>
<td>√</td>
<td>pskAPKE ($\S3$)</td>
<td>Insider-CCA &amp; Outsider-Auth ($\S4$)</td>
</tr>
</tbody>
</table>

Applications of HPKE’s PSK Modes. One class of use cases for HPKE’s PSKs is a sender transferring security guarantees from external cryptographic applications to an HPKE ciphertext. More concretely, a sender might want to transfer the post-quantum security guarantees provided by a particular PSK source to a (maybe even only otherwise classically secure) HPKE ciphertext. One such source might be PSKs distributed via an include out-of-band method (e.g., in person) or PSKs agreed upon using a post-quantum secure KEM (as demonstrated in Section 5).

In another example, HPKE’s PSKs can be used to transfer the strong authenticity guarantees of an ongoing Messaging Layer Security (MLS) group containing both sender and receiver to 1-on-1 messages between the two. MLS sessions include an HPKE (and signature) public key for each party in the group known to all other group members. A party’s signature public key is authenticated via an associated credential binding it to its owner (e.g., an X.509 certificate issued by a CA). Each user also signs their own HPKE public key to assert their ownership to rest of the group. To provide authenticity even over many years, MLS must account for signatures and HPKE keys being corrupted mid session. Rather than assuming the credentials for a corrupt signing key will be revoked, MLS instead gives users the ability to update their keys periodically (or at will). To ensure the old keys are no longer of any use, MLS also equips the group with a sequence of shared symmetric group keys. Whenever a user updates their signature or HPKE key, a new group key is produced in a way that ensures knowing the updating client’s old state (including their signature and HPKE private keys) is insufficient to learn the new group key. An MLS group including both parties $A$ and $B$ can now be used to provide strong sender authentication for $\text{HPKE}_{\text{AuthPSK}}$ ciphertexts (e.g., beyond the authenticity provided by static credentials). Say, $A$ wants to send a private message to $B$. On the one hand $A$ use their HPKE keys from the MLS session to encrypt and authenticate the message, thereby inheriting the authenticity guarantees of the credentials in the MLS session. On the other hand, $A$ can also derive a $\text{psk}$ of the current MLS group key (e.g., using MLS’s “exporter” functionality) for use with $\text{HPKE}_{\text{AuthPSK}}$. Intuitively, this provides the added guarantee to $B$ that the sender is also currently in the MLS session. The same method using $\text{HPKE}_{\text{PSK}}$ in place of $\text{HPKE}_{\text{AuthPSK}}$ gives $B$ the guarantee that the sender is a current member of the group.

1.1 Our Contributions

So far, there has only been a formal analysis of the basic mode $\text{HPKE}_{\text{Base}}$ and the authenticated mode $\text{HPKE}_{\text{Auth}}$. In this work we focus on the HPKE standard in its pre-shared key mode, both in its basic form $\text{HPKE}_{\text{PSK}}$ and in its authenticated mode
The Pre-Shared Key Modes of HPKE

Furthermore, we explore possible future post-quantum instantiations of the HPKE standard. To this end we make the following contributions.

**Pre-Shared Key (Authenticated) Public Key Encryption.** We begin, in Section 3, by introducing pre-shared key public key encryption (pskPKE) and pre-shared key authenticated public key encryption (pskAPKE) schemes, where the syntax of pskPKE matches that of the single-shot basic pre-shared mode HPKEPSK, and the syntax of pskAPKE matches that of the single-shot authenticated pre-shared mode HPKEAuthPSK. Compared to their respective non pre-shared modes PKE and APKE, encryption and decryption additionally input psk, a uniform symmetric key shared between the sender and the receiver. In terms of security, we define (active, multi-user) security notions capturing both authenticity (Auth) and privacy (CCA) for pskPKE and pskAPKE.

For pskPKE, privacy is essentially standard CCA security for PKE with the difference that the adversary additionally has access to an encryption oracle (which requires the secret pre-shared key to compute the ciphertext) and it is allowed to adaptively corrupt pre-shared (symmetric) keys and long-term (asymmetric) keys. Security holds as long as at least one of the pre-shared and long-term keys used in the challenge ciphertext/forgery has not been corrupted. Authenticity for pskPKE schemes provides the adversary with the same oracles, and the adversary’s goal is to non-trivially forge a fresh ciphertext, i.e., one that does not come from the encryption oracle.

Similar to APKE [1], for pskAPKE we consider so called weaker outsider and stronger insider security variants for privacy, and only outsider security for authenticity. Intuitively, outsider notions model settings where the adversary is an outside observer. Conversely, insider notions model settings where the adversary is somehow directly involved; in particular, even selecting some of the long-term asymmetric secrets used to produce target ciphertexts. A bit more formally, we call an honestly generated asymmetric key pair secure if the secret key was not (explicitly) leaked to the adversary and leaked if it was. An asymmetric key pair is called corrupted if it was sampled arbitrarily by the adversary. A scheme is outsider-secure if target ciphertexts are secure when produced using secure key pairs. Meanwhile, insider security holds even if one secure and one corrupted key pair are used. For example, insider privacy (Insider-CCA) for pskAPKE requires that an encapsulated key remains indistinguishable from random despite the encapsulating ciphertext being produced using corrupted sender keys (but secure receiver keys). Note that insider authenticity implies (but is stronger than) Key Compromise Impersonation (KCI) security as KCI security only requires authenticity for leaked (but not corrupt) receiver keys.

We remark that, for simplicity, our modeling assumes the pre-shared key psk to be uniformly random from some sufficiently large key-space. Indeed, The HPKE RFC mandates the PSK have at least 32 bytes of entropy to counter Partitioning Oracle Attacks [18] to which HPKE is vulnerable because it is currently only specified for AEAD schemes that are not key-committing. Thus in practice, say, hashing such a PSK to a 32 byte string prior to use with HPKE would, at least in the random oracle model, result in a near uniform distribution.

**pskKEM/DEM Construction.** In Section 4, we consider the pskKEM/DEM constructions that combine a KEM (AKEM) together with an AEAD (acting as a DEM) to obtain pskPKE (pskAPKE). First, we construct pskPKE[KEM, KS, AEAD] from a KEM, a key schedule KS, and an AEAD. To encrypt a message, a KEM ciphertext/key pair \((c, K)\) is generated.
Table 2: Security properties needed to prove Outsider-Auth and Insider-CCA security of pskAPKE obtained by the pskAKEM/DEM construction.

<table>
<thead>
<tr>
<th></th>
<th>AKEM</th>
<th>AEAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outsider</td>
<td>Outsider-Auth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Outsider-CCA</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Insider-CCA</td>
<td>X</td>
</tr>
<tr>
<td>pskAPKE</td>
<td>INT-CTXT</td>
<td>X</td>
</tr>
<tr>
<td>pskAPKE</td>
<td>IND-CPA</td>
<td>X</td>
</tr>
</tbody>
</table>

Next, the KEM key $K$ is fed together with the pre-shared key $psk$ into $KS$ to obtain the DEM key which in turn is used to AEAD-encrypt the message. At an intuitive level, the DEM key remains uniform as long as one of $K$ or $psk$ is uniform, meaning one of the receiver’s asymmetric key or the pre-shared key has not been corrupted. We will present two concrete security theorems bootstrapping privacy and authenticity of our pskPKE construction from standard security properties of the underlying KEM, KS, and AEAD. Similarly, we can construct pskAPKE[AKEM, KS, AEAD] by replacing the KEM with an AKEM in the first step of the construction above. We will again present two concrete security theorems bootstrapping privacy and authenticity of our pskAPKE construction from standard security properties of the underlying AKEM, KS, and AEAD. See also Table 2.

An Analysis of the HPKEPSK and HPKEAuthPSK Modes. Using the above mentioned transformations, the two modes HPKEPSK and HPKEAuthPSK of the HPKE standard can be obtained as pskPKE[DH-KEM, KS, AEAD] and pskAPKE[DH-AKEM, KS, AEAD]. Here DH-KEM is the well known Diffie-Hellman based KEM, DH-AKEM is the Diffie-Hellman based AKEM from [1], key schedule KS is constructed via the functions Extract and Expand both instantiated with HMAC, and AEAD is instantiated using AES-GCM or ChaCha20-Poly1305. Hence, our theorems from Section 4 provide concrete bounds for CCA and Auth security of HPKEPSK, and Insider-CCA and Outsider-Auth security of HPKEAuthPSK.

Hybrid APKE. In Section 5, we analyze a natural black-box construction of a hybrid APKE from an AKEM and from HPKEAuthPSK. Intuitively, the resulting scheme’s security depends on either one of two AKEM schemes being secure. (We remark that essentially same construction and analogous proof also construct hybrid PKE from a KEM and HPKEPSK.) One interesting application of this construction is building a PQ/classic-hybrid APKE which can be done by combining a PQ secure AKEM with a classically secure one (in HPKEAuthPSK). In comparison with the CPA secure PQ/classic-hybrid variant of HPKE in [2] our construction enjoys CCA security. Moreover, unlike [2], our construction uses both the PQ AKEM and HPKE as black-boxes meaning it can be easily implemented using only the standard interfaces to the two schemes. For these reasons our hybrid construction provides a cheap and easy path towards a practical PQ-secure (A)PKE drop-in replacement for plain HPKE.

Post-Quantum AKEM. As we have seen, AKEM schemes are the fundamental primitive underlying natural APKE and pskAPKE schemes. The HPKE standard in its HPKEAuth and HPKEAuthPSK modes relies on the Diffie-Hellman based DH-APKE instantiation. Unfortunately, none of them offers security against attackers equipped with a quantum computer.
In Section 6 we propose two generic constructions of AKEM from basic primitives that can be instantiated in a post-quantum secure way.

A well-known approach for constructing a post-quantum AKEM is to combine a post-quantum KEM with a post-quantum signature [10]. Unfortunately, the Encrypt-then-Sign (EtS) approach turns out not to be Insider-CCA secure [10]. The Sign-then-Encrypt (StE) approach is in fact Insider-CCA and Outsider-Auth secure but extending it to Sign-then-KEM in a natural way would add unnecessary overhead through the required detour over the KEM/DEM framework. Our first construction extends EtS approach to the new “Encrypt-then-Sign-then-Hash” (EtStH) approach. It combines a KEM, a digital signature SIG, and a hash function to obtain AKEM. Concretely, AKEM encryption produces a KEM ciphertext/key pair \((c,K')\) and then uses the sender’s secret key to sign ciphertext \(c\) and the sender’s public and verification key. Finally, the DEM key is derived from \(K'\), the signature and all the public keys using a hash function. Security in the sense of Insider-CCA and Outsider-Auth is proved under the assumption that the hash function is a PRF. Our scheme can be instantiated, for example, with any post-quantum secure KEM and signature scheme, for example Kyber [8] and Dilithium [11].

Our second AKEM construction relies on a non-interactive key-exchange scheme NIKE. Key encapsulation first computes an ephemeral NIKE key pair and defines the ephemeral public key as the ciphertext. Next, it derives the DEM key from the following two NIKE keys: The first (authentication) key between sender’s secret key and the receiver’s public key. The second (privacy) key between the ephemeral secret key and the receiver’s public key. Note that the knowledge of the receiver’s secret key allows to recover both NIKE keys and hence the DEM key. Security in the sense of Insider-CCA and Outsider-Auth is proved assuming the NIKE to be actively secure [13]. Instantiating it with the (actively secure) Diffie-Hellman NIKE [13], we obtain (a variant of) the DH-AKEM from the HPKE standard. But it can also be instantiated with the post-quantum secure NIKE from lattices [14] and from isogenies [12]. We remark that our NIKE approach provides deniability, whereas our more efficient EtStH construction does not.

2 Preliminaries

2.1 Notations

SETS AND ALGORITHMS. We write \(h \leftarrow S\) to denote that the variable \(h\) is uniformly sampled from the finite set \(S\). For an integer \(n\), we define \([n] := \{1,\ldots,n\}\). The notation \([b]\), where \(b\) is a boolean statement, evaluates to 1 if the statement is true and 0 otherwise.

We use uppercase letters \(A, B\) to denote algorithms. Unless otherwise stated, algorithms are probabilistic, and we write \((y_1,\ldots) \leftarrow A(x_1,\ldots)\) to denote that \(A\) returns \((y_1,\ldots)\) when run on input \((x_1,\ldots)\). We write \(A^B\) to denote that \(A\) has oracle access to \(B\) during its execution. For a randomised algorithm \(A\), we use the notation \(y \in A(x)\) to denote that \(y\) is a possible output of \(A\) on input \(x\).

SECURITY GAMES. We use standard code-based security games [6]. A game \(G\) is a probability experiment in which an adversary \(A\) interacts with an implicit challenger that answers oracle queries issued by \(A\). The game \(G\) has one main procedure and an arbitrary amount of additional oracle procedures which describe how these oracle queries are answered. We denote the (binary) output \(b\) of game \(G\) between a challenger and an
adversary $\mathcal{A}$ as $G^A \Rightarrow b$. $\mathcal{A}$ is said to win $G$ if $G^A \Rightarrow 1$, or shortly $G \Rightarrow 1$. Unless otherwise stated, the randomness in the probability term $\Pr[G^A \Rightarrow 1]$ is over all the random coins in game $G$.

We now recall definitions for (authenticated) KEMs (Section 2.2), authenticated PKE schemes (Section 2.3), and digital signatures (Section 2.6). Standard definitions of pseudo-random functions (Section 2.4), Authenticated Encryption with Associated Data (Section 2.5), and non-interactive key exchange (Section 2.7) are postponed to the appendix.

### 2.2 (Authenticated) Key Encapsulation Mechanisms

We first recall syntax and security of a KEM.

**Definition 1 (KEM).** A key encapsulation mechanism $\text{KEM}$ consists of three algorithms:

- $\text{Gen}$ outputs a key pair $(sk, pk)$, where $pk$ defines a key space $K$.
- $\text{Encaps}$ takes as input a (receiver) public key $pk$, and outputs an encapsulation $c$ and a shared secret $K \in K$ (or $\perp$).
- Deterministic $\text{Decaps}$ takes as input a (receiver) secret key $sk$ and an encapsulation $c$, and outputs a shared key $K \in K$ (or $\perp$).

We require that for all $(sk, pk) \in \text{Gen},$

$$\Pr_{(c,K) \leftarrow \text{Encaps}(pk)}[\text{Decaps}(sk,c) = K] = 1.$$

To $\text{KEM}$ we associate the two sets $SK := \{sk \mid (sk, pk) \in \text{Gen}\}$ and $PK := \{pk \mid (sk, pk) \in \text{Gen}\}$. We assume (w.l.o.g.) that there is a function $\mu : SK \rightarrow PK$ such that for all $(sk, pk) \in \text{Gen}$ it holds $\mu(sk) = pk$. We further define $PK'$ to be the set of all efficiently recognizable public keys (by $\text{Encaps}$), i.e., $PK' := \{pk \in \{0,1\}^* \mid \perp \notin \text{Encaps}(pk)\}$. Note that $PK \subseteq PK'$ by correctness, but $PK'$ could potentially contain “benign looking” public keys outside of $PK$. We will also require a property of the KEM called $\eta$-key spreadness:

$$\forall pk \in PK' : H_\infty(K \mid (c,K) \leftarrow \text{Encaps}(pk)) \geq \eta,$$

where $H_\infty$ denotes the min-entropy. This property will assure that an honestly generated key $K$ has sufficient min-entropy, even if it was generated using a $pk$ outside $PK$.

Privacy in the sense of multi-user chosen-ciphertext security is defined via the game in Listing 1. The advantage of an adversary $\mathcal{A}$ is defined as

$$\text{Adv}^{(n,q_d,q_e)}_{\mathcal{A},\text{KEM}} := \left| \Pr[(n,q_d,q_e)\text{-CCA}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right|.$$

Next, we recall syntax and security of an authenticated KEM (AKEM) [1].

**Definition 2 (AKEM).** An authenticated key encapsulation mechanism $\text{AKEM}$ consists of three algorithms:

- $\text{Gen}$ outputs a key pair $(sk, pk)$, where $pk$ defines a key space $K$.
- $\text{AuthEncap}$ takes as input a (sender) secret key $sk$ and a (receiver) public key $pk$, and outputs an encapsulation $c$ and a shared secret $K \in K$ (or $\perp$).
Listing 1: Game \((n, q_d, q_c)\)-CCA for KEM. Adversary \(A\) makes at most \(q_d\) queries to DECAP, and at most \(q_c\) queries to CHALL.

---

---

-- Deterministic AuthDecap takes as input a (sender) public key \(pk\), a (receiver) secret key \(sk\), and an encapsulation \(c\), and outputs a shared key \(K \in \mathcal{K}\) (or \(\perp\)).

We require that for all \((sk_1, pk_1) \in \text{Gen}, (sk_2, pk_2) \in \text{Gen},\)

\[
\Pr_{(c,K) \leftarrow \text{AuthEncap}(sk_1, pk_2)} [\text{AuthDecap}(pk_1, sk_2, c) = K] = 1 .
\]

Sets \(\mathcal{SK}, \mathcal{PK}, \mathcal{PK}'\), function \(\mu\), and \(\eta\)-key spreadness are defined as in the the KEM case.

Privacy (in the sense of insider and outsider CCA security) is defined via game Listing 2. Oracles AEncap and ADecap can be called with arbitrary public keys \(pk \in \mathcal{PK}' \supseteq \mathcal{PK}\), i.e., arbitrary strings that pass AKEM’s internal verification check. The insider setting is modeled using the REPSK oracle which can be used by the adversary to corrupt a sender’s secret key \(sk\). (Here we can assume \(sk \in \mathcal{SK}\) since secret keys are usually seeds and can be efficiently verified.)

Note that this security notion is equivalent to the one from [1]. In the outsider case, the adversary cannot corrupt secret keys, i.e., \(r_{sk} = 0\). The advantage is defined as

\[
\text{Adv}_{\mathcal{SK}, \mathcal{PK}', \mathcal{PK}'}^{(n, q_d, q_c, r_{sk}) - \text{Insider-CCA}} := \Pr[(n, q_c, q_d, q_c, r_{sk}) - \text{Insider-CCA}(A) \Rightarrow 1] - \frac{1}{2} ,
\]

\[
\text{Adv}_{\mathcal{SK}, \mathcal{PK}', \mathcal{PK}'}^{(n, q_d, q_c) - \text{Outsider-CCA}} := \text{Adv}_{\mathcal{SK}, \mathcal{PK}', \mathcal{PK}'}^{(n, q_d, q_c, 0) - \text{Insider-CCA}} .
\]

Authenticity is defined via the game in Listing 3.

\[
\text{Adv}_{\mathcal{SK}, \mathcal{PK}', \mathcal{PK}'}^{(n, q_d, q_c) - \text{Outsider-Auth}} := \Pr[(n, q_c, q_d) - \text{Outsider-Auth}(A) \Rightarrow 1] - \frac{1}{2} .
\]

2.3 Authenticated Public Key Encryption

We recall syntax and security of an authenticated PKE (APKE) [1].
We require that for all messages

We define the game

Privacy.

Definition 3 (APKE). An authenticated public key encryption scheme APKE consists of the following three algorithms:

- **Gen** outputs a key pair \((sk, pk)\).
- **AuthEnc** takes as input a (sender) secret key \(sk\), a (receiver) public key \(pk\), a message \(m\), associated data \(aad\), and a bitstring \(info\), and outputs a ciphertext \(c\).
- Deterministic **AuthDec** takes as input a (receiver) secret key \(sk\), a (sender) public key \(pk\), a ciphertext \(c\), associated data \(aad\) and a bitstring \(info\), and outputs a message \(m\).

We require that for all messages \(m \in \{0,1\}^*, aad \in \{0,1\}^*, info \in \{0,1\}^*\),

\[
\Pr_{(sk_S, pk_R) \get \text{Gen}} \left[ c \leftarrow \text{AuthEnc}(sk_S, pk_R, m, aad, info) \right] = 1 .
\]

Sets \(SK\), \(PK\), \(PK'\), function \(\mu\), and \(\eta\)-key spreadness are defined as in the the KEM case.

Privacy. We define the game \((n, q_e, q_d, q_e)\)-Insider-CCA in Listing 4, which is the strongest privacy notion for APKE defined in [1].

The advantage of \(A\) is

\[
\text{Adv}_{A}^{(n, q_e, q_d, q_e)\text{-Insider-CCA}} := \left| \Pr[(n, q_e, q_d, q_e)\text{-Insider-CCA}(A) \Rightarrow 1] - \frac{1}{2} \right| .
\]
Listing 3: Game \((n, q_e, q_d)\)-Outsider-Auth for AKEM. Adversary \(A\) makes at most \(q_e\) queries to \(\text{AEncap}\), and at most \(q_d\) queries to \(\text{ADecap}\).

\[
\begin{align*}
(n, q_e, q_d)\text{-Outsider-Auth} & \\
01 & \text{for } i \in [n] \\
02 & (sk_i, pk_i) \xleftarrow{\$} \text{Gen} \\
03 & \mathcal{E} \leftarrow \emptyset \\
04 & b \xleftarrow{\$} \{0, 1\} \\
05 & b' \xleftarrow{\$} A^{\text{AEncap}, \text{ADecap}}(pk_1, \ldots, pk_n) \\
06 & \text{return } [b = b']
\end{align*}
\]

Oracle \(\text{AEncap}(i \in [n], pk \in \mathcal{PK}')\)

\[
\begin{align*}
07 & (c, K) \xleftarrow{\$} \text{AuthEncap}(sk_i, pk) \\
08 & \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, K)\} \\
09 & \text{return } (c, K)
\end{align*}
\]

Oracle \(\text{ADecap}(pk \in \mathcal{PK}', j \in [n], c)\)

\[
\begin{align*}
10 & \text{if } \exists K : (pk, pk_j, c, K) \in \mathcal{E} \\
11 & \text{return } K \\
12 & K \xleftarrow{\$} \text{AuthDecap}(pk, sk_j, c) \\
13 & \text{if } b = 1 \land pk \in \{pk_1, \ldots, pk_n\} \land K \neq \bot \\
14 & K \xleftarrow{\$} K \\
15 & \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, c, K)\} \\
16 & \text{return } K
\end{align*}
\]

AUTHENTICITY. Furthermore, in Listing 5 we recall the \((n, q_e, q_d)\)-Outsider-Auth game from [1]. The advantage of \(A\) is defined as

\[
\text{Adv}_{A, \text{APKE}}^{(n, q_e, q_d)\text{-Outsider-Auth}} := \text{Pr}\left[(n, q_e, q_d)\text{-Outsider-Auth} \Rightarrow 1\right].
\]

Note that in contrast to the privacy case we use the weaker outsider notion instead of the insider notion for authenticity. This is because [1] show that the HPKE_{\text{Auth}} construction cannot fulfill insider authenticity for any possible instantiation. Since the same attack can be run against HPKE_{\text{PSK}}, we omit the definition here.

2.4 Pseudorandom Functions

A keyed function \(F\) with a finite key space \(\mathcal{K}\), input length \(n\), and a finite output range \(\mathcal{R}\) is a function \(F : \mathcal{K} \times \{0, 1\}^* \rightarrow \mathcal{R}\).

Definition 4 (Multi-Instance Pseudorandom Function). The \((n_k, q_{\text{PRF}})\)-PRF advantage of an adversary \(A\) against a keyed function \(F\) with finite key space \(\mathcal{K}\), and finite range \(\mathcal{R}\) is defined as

\[
\text{Adv}_{A, F}^{(n_k, q_{\text{PRF}})\text{-PRF}} := \left| \text{Pr}_{K_1, \ldots, K_{n_k} \xleftarrow{\$} \mathcal{K}} \left[ A^{F(K_1, \cdot), \ldots, F(K_{n_k}, \cdot)} \right] - \text{Pr}_{f_1(\cdot), \ldots, f_{n_k}(\cdot)} A^{f_1(\cdot), \ldots, f_{n_k}(\cdot)} \right|,
\]

where \(f_i : \{0, 1\}^* \rightarrow \mathcal{R}\) for \(i \in [n_k]\) are chosen uniformly at random from the set of functions mapping to \(\mathcal{R}\) and \(A\) makes at most \(q_{\text{PRF}}\) queries in total to the oracles \(F(K_i, \cdot)\), \(f_i\) resp.

Definition 5 (2-Keyed Function). A 2-keyed function \(F\) with finite key spaces \(\mathcal{K}_1\) and \(\mathcal{K}_2\), and finite range \(\mathcal{R}\) is a function

\[
F : \mathcal{K}_1 \times \mathcal{K}_2 \times \{0, 1\}^* \rightarrow \mathcal{R}.
\]
Listing 4: Game \((n, q_c, q_d, q_e)\)-Insider-CCA for APKE in which adversary \(A\) makes at most \(q_c\) queries to \(\text{AEnc}\), at most \(q_d\) queries to \(\text{ADec}\) and at most \(q_e\) queries to \(\text{CHALL}\).

\[
\text{(n, q_c, q_d, q_e)-Insider-CCA} \\
\text{for } i \in [n] \\
\text{if } (s_k, p_k) \in \text{Gen} \\
\text{oracle } A^\text{Oracle} = \emptyset \\
\text{b' } \in \{0,1\} \\
\text{return } [b = b'] \\
\text{oracle } \text{AEnc}(i \in [n], p_k \in \mathcal{K}', m, \text{aad, info}) \\
\text{c } \in \text{AuthEnc}(sk, p_k, m, \text{aad, info}) \\
\text{return } c
\]

\[
\text{Oracle } \text{ADec}(pk \in \mathcal{K}', j \in [n], c, \text{aad, info}) \\
\text{if } (pk, p_k, c, \text{aad, info}) \in \mathcal{E} \\
\text{return } \perp \\
\text{m } \leftarrow \text{AuthDec}(pk, sk, c, \text{aad, info}) \\
\text{return } m
\]

\[
\text{Oracle } \text{CHALL}(sk \in \mathcal{K}, j \in [n], m_0, m_1, \text{aad, info}) \\
\text{if } |m_0| \neq |m_1| \text{ return } \perp \\
\text{c } \leftarrow \text{AuthEnc}(sk, pk, m, \text{aad, info}) \\
\text{E } \leftarrow \mathcal{E} \cup \{\mu(s_k, p_k, c, \text{aad, info})\} \\
\text{return } c
\]

Listing 5: Game \((n, q_c, q_d)\)-Outsider-Auth for APKE in which adversary \(A\) makes at most \(q_c\) queries to \(\text{AEnc}\) and at most \(q_d\) queries to \(\text{ADec}\).

\[
\text{(n, q_c, q_d)-Outsider-Auth} \\
\text{for } i \in [n] \\
\text{if } (sk, p_k) \in \text{Gen} \\
\text{oracle } A^\text{Oracle} = \emptyset \\
\text{c' } \in \{0,1\} \\
\text{return } c \\
\text{oracle } \text{AEnc}(i \in [n], p_k \in \mathcal{K}', m, \text{aad, info}) \\
\text{c } \in \text{AuthEnc}(sk, p_k, m, \text{aad, info}) \\
\text{E } \leftarrow \mathcal{E} \cup \{\mu(s_k, p_k, c, \text{aad, info})\} \\
\text{oracle } \text{ADec}(pk \in \mathcal{K}', j \in [n], c, \text{aad, info}) \\
\text{m } \leftarrow \text{AuthDec}(sk, p_k, m, \text{aad, info}) \\
\text{return } m
\]

2.5 Authenticated Encryption with Associated Data

We recall standard syntax and security for AEAD schemes.

**Definition 6 (AEAD).** A nonce-based authenticated encryption scheme with associated data and key space \(\mathcal{K}'\) consists of the following two algorithms:

- Deterministic algorithm \(\text{AEAD.Enc}\) takes as input a key \(k \in \mathcal{K}'\), a message \(m\), associated data \(aad\), and a nonce and outputs a ciphertext \(c\).
- Deterministic algorithm \(\text{AEAD.Dec}\) takes as input a key \(k \in \mathcal{K}'\), a ciphertext \(c\), associated data \(aad\) and a nonce \(nonce\) and outputs a message \(m\) or the failure symbol \(\perp\).

We require that for all \(aad \in \{0,1\}^*\), \(m \in \{0,1\}^*\), \(nonce \in \{0,1\}^{N_{nonce}}\)

\[
\Pr_{k \in \mathcal{K}'}[[\text{AEAD.Dec}(k, \text{AEAD.Enc}(k, m, aad, nonce), aad, nonce) = m]] = 1
\]

where \(N_{nonce}\) is the length of the nonce in bits.

We define the multi-instance security game \((n, q_d)\)-\text{INT-CTX} in Listing 6 and \((n, q_d)\)-CCA in Listing 7. Note that an AEAD scheme which is IND-CPA and INT-CTX secure is also CCA secure [5]. The advantage of an adversary \(A\) is

\[
\text{Adv}_{A, \text{AEAD}}^{(n, q_d)\text{-INT-CTX}} := \left| \Pr[(n, q_d)\text{-INT-CTX}(A) \Rightarrow 1] - \frac{1}{2} \right|
\]

\[
\text{Adv}_{A, \text{AEAD}}^{(n, q_d)\text{-CCA}} := \left| \Pr[(n, q_d)\text{-CCA}(A) \Rightarrow 1] - \frac{1}{2} \right|
\]
The signature scheme fulfills correctness if for all \( (\text{SIG}, \text{Vfy}) \in \text{SIG} \) it holds
\[
\Pr_{\sigma \leftarrow \text{Sign}(\text{sk}, m)} [\text{Vfy}(\text{vk}, m, \sigma) = 1] = 1.
\]
To \( \text{SIG} \) we associate the two sets \( \mathcal{SK} := \{ \text{sigk} \mid (\text{sigk}, \text{vk}) \in \text{Gen} \} \) and \( \mathcal{VK} := \{ \text{vk} \mid (\text{sigk}, \text{vk}) \in \text{Gen} \} \). We assume (w.l.o.g.) that there is a function \( \mu' : \mathcal{SK} \rightarrow \mathcal{VK} \) such that for all \( (\text{sigk}, \text{vk}) \in \text{Gen} \) it holds \( \mu'(\text{sigk}) = \text{vk} \).

**Definition 8 (Strong Unforgeability).** Let \( \text{SIG} = (\text{Gen}, \text{Sign}, \text{Vfy}) \) be a signature scheme. We define multi-user strong unforgeability against a chosen message attack

\[
\text{Game}((n, q_d)-\text{CCA}) \text{ for AEAD. Adversary } \mathcal{A} \text{ makes at most one query per index } \iota \text{ to } \text{Enc} \text{ and at most } q_d \text{ queries in total to } \text{Dec}.
\]

\[
\begin{array}{|c|c|}
\hline
\text{(n,qd)-INT-CTX} & \text{Oracle ENC}(\iota \in [n], m, \text{aad}) \\
01 \text{ for } \iota \in [n] & 08 \text{ c} \leftarrow \text{AEAD.Enc}(k, m, \text{aad}, \text{nonce}_i) \\
02 k_i \leftarrow K_i & 09 \mathcal{E} \leftarrow \mathcal{E} \cup \{(i, m, c, \text{aad})\} \\
03 \text{ nonce}_i \leftarrow \{0,1\}^{\text{nonce}} & 10 \text{ return } (c, \text{nonce}_i) \\
04 \mathcal{E} \leftarrow \emptyset & 11 \text{ if } b = 0 \\
05 b, c \leftarrow \{0,1\} & 12 m \leftarrow \text{AEAD.Dec}(k, c, \text{aad}, \text{ nonce}_i) \\
06 b' \leftarrow \mathcal{A}^{\text{Enc}, \text{Dec}} & 13 \text{ else if } \exists m' : (i, m', c, \text{aad}) \in \mathcal{E} \\
07 \text{ return } [b = b'] & 14 m \leftarrow m' \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{(n,qd)-CCA} & \text{Oracle DEC}(\iota \in [n], c, \text{aad}) \\
01 \text{ for } \iota \in [n] & 08 \text{ c} \leftarrow \text{AEAD.Enc}(k, m, \text{aad}, \text{ nonce}_i) \\
02 k_i \leftarrow K_i & 09 \mathcal{E} \leftarrow \mathcal{E} \cup \{(i, c, \text{aad})\} \\
03 \text{ nonce}_i \leftarrow \{0,1\}^{\text{nonce}} & 10 \text{ return } (c, \text{nonce}_i) \\
04 \mathcal{E} \leftarrow \emptyset & 11 \text{ if } (i, c, \text{aad}) \in \mathcal{E} \\
05 b, c \leftarrow \{0,1\} & 12 \text{ return } 1 \\
06 b' \leftarrow \mathcal{A}^{\text{Enc}, \text{Dec}} & 13 m \leftarrow \text{AEAD.Dec}(k, c, \text{aad}, \text{ nonce}_i) \\
07 \text{ return } [b = b'] & 14 \text{ return } m \\
\hline
\end{array}
\]

**2.6 Digital Signatures**

**Definition 7 (Signature Scheme).** A signature scheme \( \text{SIG} = (\text{Gen}, \text{Sign}, \text{Vfy}) \) consists of three algorithms:

- **Key generation** \( \text{Gen} \) generates a secret signing key \( \text{sigk} \) and a verification key \( \text{vk} \).
- **Signing** \( \text{Sign} \): On input a signing key \( \text{sigk} \) and a message \( m \), outputs a signature \( \sigma \).
- **Verification** \( \text{Vfy} \): On input a verification key \( \text{vk} \) and a message \( m \), and a signature \( \sigma \), deterministically outputs a bit \( b \).

The signature scheme fulfills correctness if for all \( (\text{sigk}, \text{vk}) \leftarrow \text{Gen} \) it holds
\[
\Pr_{\sigma \leftarrow \text{Sign}(\text{sk}, m)} [\text{Vfy}(\text{vk}, m, \sigma) = 1] = 1.
\]
(SU−CMA) via the (n,q_s)-SU−CMA game in Listing 8. The advantage of an adversary \(A\) is 
\[\text{Adv}_{(n,q_s)-SU−CMA}^A = Pr[(n,q_s)-SU−CMA(A) \Rightarrow 1].\]

Listing 8: Game (n,q_s)-SU−CMA for SIG. Adversary \(A\) makes at most \(q_s\) queries to 
\(\text{SIGN}\).

\[
\begin{array}{|c|}
\hline
(n,q_s)-SU−CMA \quad \text{Oracle SIGN}(i \in [n],m) \\
\hline
01 \text{ for } i \in [n] \\
02 \quad (\text{sig}_i, vk_i) \xleftarrow{} \text{Gen} \\
03 \quad Q \leftarrow \emptyset \\
04 \quad (t^*, m^*, \sigma^*) \xleftarrow{} A^{\text{SIGN}}(vk_1, \ldots, vk_n) \\
05 \quad \text{return } [\text{Vfy}(vk_i^*, m^*, \sigma^*) = 1 \\
06 \quad \wedge (t^*, m^*, \sigma^*) \notin Q] \\
07 \quad \sigma \xleftarrow{} \text{SIGN}(\text{sig}_i, m) \\
08 \quad Q \leftarrow Q \cup \{(i,m,\sigma)\} \\
09 \quad \text{return } \sigma \\
\hline
\end{array}
\]

2.7 Non-Interactive Key Exchange

Definition 9 (Non-Interactive Key Exchange). A NIKE scheme consists of a setup, 
two algorithms NIKE.KeyGen, NIKE.SharedKey and a shared key space \(SHK\). The algorithms 
are defined as follows:

- \(\text{NIKE.KeyGen}\) outputs a pair of public and secret key \((sk, pk)\).
- \(\text{NIKE.SharedKey}\) takes a secret key \(sk\) and a public key \(pk\) and outputs either a shared 
  key in \(SHK\) or the failure symbol \(\bot\).

The NIKE fulfills correctness if for all \((sk_1, pk_1) \xleftarrow{} \text{NIKE.KeyGen}, (sk_2, pk_2) \xleftarrow{} \text{NIKE.KeyGen}\), it holds

\[\text{NIKE.SharedKey}(sk_1, pk_2) = \text{NIKE.SharedKey}(sk_2, pk_1).\]

Sets \(SK, PK, PK'_1\), and function \(\mu\) are defined as in the the KEM case. We define the 
entropy of the public key as

\[P_{\text{NIKE}} := \max_{pk} \Pr[pk = pk' \mid (sk', pk') \xleftarrow{} \text{NIKE.KeyGen}].\]

Note that this statistical property can be upper bounded by the passive security of NIKE.

The security definition for actively secure NIKE can be found in Appendix A.1.

3 Pre-Shared Key (Authenticated) Encryption

In this section, we define syntax and security of pre-shared key public key encryption 
(\(p_{skPKE}\)) and pre-shared key authenticated public key encryption (\(p_{skAPKE}\)). The former 
is an extension of common public key encryption with an additional pre-shared symmetric 
key that has already been shared between the parties. The latter is an analogue extension 
of authenticated public key encryption (APKE). The pre-shared key (\(psk\)) has two 
 functionalities. First, it provides an additional layer of privacy since the security does 
not have to rely on the asymmetric key only. Second, it also provides authenticity. The
The intuition behind security is that even if one of the two keys, either the asymmetric key or the (symmetric) pre-shared key, is corrupted or in any other way insecure, the scheme should still guarantee security.

We start with defining the syntax of \texttt{pskPKE} and \texttt{pskAPKE} in Section 3.1 and then define the security model for privacy (Section 3.2) and authenticity (Section 3.3).

### 3.1 Syntax

**Definition 10 (pskPKE).** A pre-shared key public key encryption scheme \texttt{pskPKE} consists of the following four algorithms:

- \texttt{GenSK} outputs a key pair \((sk, pk)\).
- \texttt{GenPSK} outputs a pre-shared key \(psk\).
- \texttt{pskEnc} takes as input a (receiver) public key \(pk\), a pre-shared key \(psk\), a message \(m\), associated data \(aad\), and outputs a ciphertext \(c\).
- Deterministic \texttt{pskDec} takes as input a (receiver) secret key \(sk\), a pre-shared key \(psk\), a ciphertext \(c\), associated data \(aad\) and a bitstring \(info\), and outputs a message \(m\).

We require that for all messages \(m \in \{0,1\}^*\), \(aad \in \{0,1\}^*\), \(info \in \{0,1\}^*\),

\[
\Pr_{(sk,pk) \leftarrow \text{GenSK}, psk \leftarrow \text{GenPSK}} [c \leftarrow \text{pskEnc}(pk, psk, m, aad, info), \text{pskDec}(sk, psk, c, aad, info) = m] = 1.
\]

**Definition 11 (pskAPKE).** A pre-shared key Authenticated public key encryption scheme \texttt{pskAPKE} consists of the following four algorithms:

- \texttt{GenSK} outputs a key pair \((sk, pk)\).
- \texttt{GenPSK} outputs a pre-shared key \(psk\).
- \texttt{pskAEnc} takes as input a (sender) secret key \(sk\), a (receiver) public key \(pk\), a pre-shared key \(psk\), a message \(m\), associated data \(aad\), a bitstring \(info\), and outputs a ciphertext \(c\).
- Deterministic \texttt{pskADec} takes as input a (sender) public key \(pk\), a (receiver) secret key \(sk\), a pre-shared key \(psk\), a ciphertext \(c\), associated data \(aad\) and a bitstring \(info\), and outputs a message \(m\).

We require that for all messages \(m \in \{0,1\}^*\), \(aad \in \{0,1\}^*\), \(info \in \{0,1\}^*\),

\[
\Pr_{(sk_S, pk_R) \leftarrow \text{GenSK}, (sk_R, pk_S) \leftarrow \text{GenSK}} [c \leftarrow \text{pskAEnc}(sk_S, pk_R, psk, m, aad, info), \text{pskADec}(pk_S, sk_R, psk, c, aad, info) = m] = 1.
\]

Sets \(SK\), \(PK\), \(PK'\), and function \(\mu\) are defined as in the \texttt{KEM} case.
Listing 9: Game \( (n, q_c, q_d, q_e, r_{pk}, r_{psk}) \)-CCA for \( pskPKE \). Adversary \( A \) makes at most \( q_c \) queries to \( ENC \), at most \( q_d \) queries to \( DEC \), at most \( q_e \) queries to \( CHALL \), at most \( r_{pk} \) queries to \( REPPK \), and at most \( r_{psk} \) queries to \( REPSK \).

\[
\begin{array}{ll}
(n, q_c, q_d, q_e, r_{pk}, r_{psk})-\text{CCA} & \\
01 & \text{for } i \in [n] \\
02 & (sk_i, pk_i) \leftarrow \text{Gen} \\
03 & \text{for } j \in [n] \\
04 & psk_{ij} \leftarrow \text{GenPSK} \\
05 & psk_{ij} \leftarrow psk_{ij} \\
06 & E, \Gamma_{pk}, \Gamma_{psk} \leftarrow \emptyset \\
07 & b \leftarrow \{0,1\} \\
08 & b' \leftarrow A^{Enc,Dec,Chall,Reppk,Repsk}(pk_1,\ldots,pk_n) \\
09 & \text{return } [b = b']
\end{array}
\]

Oracle \( CHALL(i \in [n], j \in [n], m_0, m_1, \text{aad, info}) \)
\[
\begin{array}{ll}
10 & \text{if } [m_0] \neq [m_1] \lor (j \in I_{pk} \lor (i,j) \in I_{psk}) \\
11 & \text{return } \bot \\
12 & c \leftarrow \text{pskEnc}(pk_j, psk_{ij}, m_0, \text{aad, info}) \\
13 & \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, psk_{ij}, c, \text{aad, info})\} \\
14 & \text{return } c
\end{array}
\]

Oracle \( \text{Reppk}(j \in [n], pk \in \mathcal{P}^c) \)
\[
\begin{array}{ll}
15 & (sk_j, pk_j) \leftarrow (\bot, pk) \\
16 & \Gamma_{pk} \leftarrow \Gamma_{pk} \cup \{j\}
\end{array}
\]

Oracle \( \text{Repsk}(i \in [n], j \in [n], pk) \)
\[
\begin{array}{ll}
17 & psk_{ij} \leftarrow psk \\
18 & psk_{ij} \leftarrow psk \\
19 & \Gamma_{psk} \leftarrow \Gamma_{psk} \cup \{(i,j), (j,i)\}
\end{array}
\]

\[
\begin{array}{ll}
(n, q_c, q_d, q_e, r_{pk}, r_{psk})-\text{Insider-CCA} & \\
01 & \text{for } i \in [n] \\
02 & (sk_i, pk_i) \leftarrow \text{Gen} \\
03 & \text{for } j \in [i] \\
04 & psk_{ij} \leftarrow \text{GenPSK} \\
05 & psk_{ij} \leftarrow psk_{ij} \\
06 & E, \Gamma_{pk}, \Gamma_{psk} \leftarrow \emptyset \\
07 & b \leftarrow \{0,1\} \\
08 & b' \leftarrow A^{Enc,Dec,Chall,Reppk,Repsk}(pk_1,\ldots,pk_n) \\
09 & \text{return } [b = b']
\end{array}
\]

Oracle \( CHALL(i \in [n], j \in [n], m_0, m_1, \text{aad, info}) \)
\[
\begin{array}{ll}
10 & \text{if } [m_0] \neq [m_1] \lor (j \in \Gamma_{pk} \lor (i,j) \in \Gamma_{psk}) \\
11 & \text{return } \bot \\
12 & c \leftarrow \text{pskAEnc}(sk_i, pk_j, psk_{ij}, m_0, \text{aad, info}) \\
13 & \mathcal{E} \leftarrow \mathcal{E} \cup \{(sk_i, pk_j, psk_{ij}, c, \text{aad, info})\} \\
14 & \text{return } c
\end{array}
\]

Oracle \( \text{Reppk}(j \in [n], pk \in \mathcal{P}^c) \)
\[
\begin{array}{ll}
15 & (sk_j, pk_j) \leftarrow (\bot, pk) \\
16 & \Gamma_{pk} \leftarrow \Gamma_{pk} \cup \{j\}
\end{array}
\]

Oracle \( \text{Repsk}(i \in [n], j \in [n], pk) \)
\[
\begin{array}{ll}
17 & psk_{ij} \leftarrow psk \\
18 & psk_{ij} \leftarrow psk \\
19 & \Gamma_{psk} \leftarrow \Gamma_{psk} \cup \{(i,j), (j,i)\}
\end{array}
\]
Listing 11: Game \((n, q_c, q_d, r_{pk}, r_{psk})\)-Auth for pskPKE. Adversary \(\mathcal{A}\) makes at most \(q_c\) queries to \(\text{ENC}\), at most \(q_d\) queries to \(\text{DEC}\), at most \(r_{pk}\) queries to \(\text{RepPK}\), and at most \(r_{psk}\) queries to \(\text{RepPSK}\).

\[
\begin{array}{ll}
(n, q_c, q_d, r_{pk}, r_{psk})\text{-Auth} & \text{Oracle Dec}(i \in [n], j \in [n], c, aad, info) \\
01 & \text{for } i \in [n] \\
02 & (sk_i, pk_i) \triangleq \text{Gen} \\
03 & \text{for } j \in [i] \\
04 & \text{psk}_{ij} \triangleq \text{GenPSK} \\
05 & \text{psk}_{ji} \leftarrow \text{psk}_{ij} \\
06 & E, \Gamma_{\text{psk}} \leftarrow \emptyset \\
07 & ((i^*, j^*, c^*, aad^*, info^*) \in A^{\text{Enc, Dec, RepPK, RepPSK}}(pk_1, \ldots, pk_n)) \\
08 & \text{return } [\text{if } (i^*, j^*) \not\in \Gamma_{\text{psk}} \land sk_{ij} \neq \perp \\
& \land (pk_{ij}, \text{psk}_{ij}, c^*, aad^*, info^*) \notin E \\
& \land \text{pskDec}(sk_{ij}, \text{psk}_{ij}, c^*, aad^*, info^*) \neq \perp [)] \\
\end{array}
\]

Oracle \(\text{Enc}(i \in [n], j \in [n], m, aad, info)\)

\[
09 & c \triangleq \text{Enc}(pk_j, \text{psk}_{ij}, m, aad, info) \\
10 & E \leftarrow E \cup \{(pk_j, \text{psk}_{ij}, c, aad, info)\} \\
11 & \text{return } c
\]

Listing 12: Game \((n, q_c, q_d, r_{pk}, r_{psk})\)-Outsider-Auth for pskAPKE. Adversary \(\mathcal{A}\) makes at most \(q_c\) queries to \(\text{ENC}\), at most \(q_d\) queries to \(\text{DEC}\), at most \(r_{pk}\) queries to \(\text{RepPK}\), and at most \(r_{psk}\) queries to \(\text{RepPSK}\).

\[
\begin{array}{ll}
(n, q_c, q_d, r_{pk}, r_{psk})\text{-Outsider-Auth} & \text{Oracle Dec}(i \in [n], j \in [n], c, aad, info) \\
01 & \text{for } i \in [n] \\
02 & (sk_i, pk_i) \triangleq \text{Gen} \\
03 & \text{for } j \in [i] \\
04 & \text{psk}_{ij} \triangleq \text{GenPSK} \\
05 & \text{psk}_{ji} \leftarrow \text{psk}_{ij} \\
06 & E, \Gamma_{\text{psk}} \Gamma_{\text{pk}} \leftarrow \emptyset \\
07 & ((i^*, j^*, c^*, aad^*, info^*) \in A^{\text{Enc, Dec, RepPK, RepPSK}}(pk_1, \ldots, pk_n)) \\
08 & \text{return } [\text{if } (i^* \notin \Gamma_{\text{pk}} \lor (i^*, j^*) \notin \Gamma_{\text{psk}}) \land sk_{ij} \neq \perp \\
& \land (pk_{ij}, \text{psk}_{ij}, c^*, aad^*, info^*) \notin E \\
& \land \text{pskAdDec}(pk_{ij}, \text{sk}_{ij}, \text{psk}_{ij}, c^*, aad^*, info^*) \neq \perp [)] \\
\end{array}
\]

Oracle \(\text{Enc}(i \in [n], j \in [n], m, aad, info)\)

\[
09 & \text{if } sk_j = \perp \\
10 & \text{return } \perp \\
11 & c \triangleq \text{pskAdEnc}(sk_i, pk_j, \text{psk}_{ij}, m, aad, info) \\
12 & E \leftarrow E \cup \{(pk_i, \text{psk}_{ij}, c, aad, info)\} \\
13 & \text{return } c
\]

The Pre-Shared Key Modes of HPKE
3.2 Privacy

Privacy is defined via the games in Listing 9 (pskPKE) and Listing 10 (pskAPKE). The idea is based on the standard CCA definition for PKE with the following modifications.

For the game in Listing 9, the adversary is provided with an encryption oracle $Enc$ since, in contrast to standard PKE, encryption requires the knowledge of the corresponding pre-shared key. We further strengthen the security model by allowing corrupted keys. This is modeled by two oracles, $RepPK$ and $RepPSK$. Oracle $RepPK$ models the corruption of an asymmetric key pair, where the adversary is allowed to replace the public key of a user with $pk \in PK'$ (with or without knowing the matching private key). Since the game is not able to decrypt queries to a corrupted receiver’s key anymore, we return ⊥ on such queries. To keep track of corrupted keys, the game maintains a set $I_{pk}$ containing all corrupted indices $j$. Similarly, oracle $RepPSK$ models the corruption of a pre-shared key between two chosen users. Set $I_{psk}$ keeps track of sender and receiver pairs for which the corresponding $psk$ was corrupted. Provided with the additional oracles, it should still be hard to guess the challenge bit, i.e., which of the two messages given to the challenge oracle was encrypted. To avoid trivial wins, we disallow the challenge oracle to be queried on pairs of users for which both the receiver’s key as well as the pre-shared key were corrupted. However, the challenge query is allowed if at most one of them is corrupted. (Note that in particular the adversary is still allowed to issue challenge queries for a corrupted $psk$.) In that sense we model an “insider setting”. An insider notion for pskPKE cannot be formulated stronger because it does not make any sense to allow corrupted senders in the sense of a corrupted $sk$ since it is not used for encryption.\(^5\) The advantage of adversary $A$ is

$$Adv_{A, pskPKE}^{(n,q_e,q_d,q_c,r_{pk},r_{psk})-CCA} := \Pr[(n,q_e,q_d,q_c,r_{pk},r_{psk})-CCA(A) \Rightarrow 1] - \frac{1}{2}.$$

The definition of CCA security for pskAPKE (Listing 10) works similar with the following changes. The main difference compared to pskPKE is that the asymmetric part of the encryption is authenticated which means that the sender’s secret key is also involved. To take attacks into account which could make use of corrupted senders, we make the following modification. The game provides another oracle $RepSK$ which allows the adversary to replace an asymmetric secret key directly, which means they are also allowed to query all the other oracles on corrupted sender keys. This is exactly what is called the “Insider” setting in [1] and signcryption [10]. This means in particular that the encryption of two different messages is indistinguishable even if the sender’s asymmetric key was adversarially chosen. As in the case of pskPKE, the $psk$ can also be corrupted in the sense of an insider attack, i.e., the sender is corrupted. However, due to the symmetric nature of the pre-shared key, this implies a security loss for the receiver as well and we have to exclude trivial wins in the same way as for pskPKE. Therefore, the same security requirement as for pskPKE holds, i.e., in addition to the corruption of a sender’s secret key either a receiver’s key or the pre-shared key can be corrupted and security still holds. We also define outsider security as the simplified setting, where the adversary does not have access to the $RepSK$

\(^5\) To prevent confusion we do explicitly call this notion insider secure and only use the term “insider” if it is possible to the (asymmetric) secret key of a sender. See also the security definition of the pskAPKE.
oracle, i.e., $r_{pk} = 0$. The advantage of adversary $\mathcal{A}$ is

$$\text{Adv}_{\mathcal{A},\text{pskAPKE}}^{(n, q_e, q_d, q_c, r_{pk}, r_{psk})-\text{Insider-CCA}} := \Pr\left[ (n, q_e, q_d, q_c, r_{pk}, r_{psk})\Rightarrow 1 \right] - \frac{1}{2},$$

$$\text{Adv}_{\mathcal{A},\text{pskAPKE}}^{(n, q_e, q_d, q_c, r_{pk}, r_{psk})-\text{Outsider-CCA}} := \text{Adv}_{\mathcal{A},\text{pskAPKE}}^{(n, q_e, q_d, q_c, r_{pk}, r_{psk})-\text{Insider-CCA}}.$$

### 3.3 Authenticity

Authenticity is defined via the games in Listing 11 (pskPKE) and Listing 12 (pskAPKE). For the game in Listing 11, the goal is to forge a fresh ciphertext, i.e., one that was not output by the encryption oracle and that does not decrypt to $\bot$. Further, we do not allow corrupted receiver keys for the challenge, i.e. $sk_j \neq \bot$, since the decryption would not be possible anymore. As in the privacy case, we model corrupted keys via oracle RepPK which allows the adversary to replace an asymmetric key and oracle RepPSK which allows to replace a pre-shared key. Due to the structure of a pskPKE, authenticity cannot rely on the asymmetric keys since the sender needs no secret material for the encryption. Hence, authenticity can only be achieved via the pre-shared key. To avoid trivial wins, the game excludes forgeries for which the corresponding pre-shared key was corrupted, i.e., replaced via oracle RepPSK. This means, an adversary wins if they can forge a new valid ciphertext for which the pre-shared key was not corrupted given encryption and decryption oracles as well corruption oracles for both the keys. The advantage of adversary $\mathcal{A}$ is

$$\text{Adv}_{\mathcal{A},\text{pskPKE}}^{(n, q_e, q_d, r_{pk}, r_{psk})-\text{Auth}} := \Pr\left[ (n, q_e, q_d, r_{pk}, r_{psk})\Rightarrow 1 \right].$$

For the Auth security of a pskAPKE (Listing 12) there is only a slight modification. Encryption pskAEnc also inputs the asymmetric sender’s secret key, which is why the authenticity can now also rely on the sender’s asymmetric key and not only on the pre-shared key. In the game, this is used for a stronger notion which considers corruptions of the sender’s key and corruptions of the pre-shared key. The adversary’s forgery is accepted if at most one of the keys was corrupted. This means, it should be hard to forge a fresh ciphertext even if the sender’s key or the pre-shared key were corrupted but not both. The advantage of adversary $\mathcal{A}$ is

$$\text{Adv}_{\mathcal{A},\text{pskAPKE}}^{(n, q_e, q_d, r_{pk}, r_{psk})-\text{Outsider-Auth}} := \Pr\left[ (n, q_e, q_d, r_{pk}, r_{psk})\Rightarrow 1 \right].$$

One could also define Insider-Auth security for pskAPKE by allowing the adversary to choose the secret key of the receiver of a forgery (and adjust the non-triviality condition accordingly). Since we do not use Insider-Auth for pskAPKE in the following, we omit the formal definition.

### 4 HPKE’s constructions of a pskPKE and pskAPKE

#### 4.1 Generic Constructions

We construct a pskPKE and a pskAPKE from an (authenticated) key encapsulation mechanism KEM (AKEM), a 2-keyed function KS (where KS$_1$ denotes KS keyed in the first input
exists an $D$-KEM. More concretely, that is $\text{pskPKE} \{ \text{KEM}, \text{KS}, \text{AEAD} \}$ where $\text{pskPKE} \text{GenSK} = \text{KEM} \text{Gen}$, and the pre-shared key generation $\text{pskPKE} \text{GenPSK}$ samples a uniformly random element from the appropriate key space. Encryption and decryption are defined in Listing 13 and Listing 14, respectively.

**Listing 13:** Encryption and decryption functions of the pre-shared-key PKE scheme $\text{pskPKE} \{ \text{KEM}, \text{KS}, \text{AEAD} \}$, built from KEM, KS, and AEAD.

<table>
<thead>
<tr>
<th>$\text{pskEnc}(pk, psk, m, aad, info)$</th>
<th>$\text{pskDec}(sk, psk, (c_1, c_2), aad, info)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 $(c_1, K) \leftarrow \text{Encaps}(pk)$</td>
<td>05 $K \leftarrow \text{Decaps}(sk, c_1)$</td>
</tr>
<tr>
<td>02 $(k, \text{nonce}) \leftarrow \text{KS}(K, psk, info)$</td>
<td>06 $(k, \text{nonce}) \leftarrow \text{KS}(K, psk, info)$</td>
</tr>
<tr>
<td>03 $c_2 \leftarrow \text{AEAD.Enc}(k, m, aad, nonce)$</td>
<td>07 $m \leftarrow \text{AEAD.Dec}(k, c_2, aad, nonce)$</td>
</tr>
<tr>
<td>04 $\text{return } (c_1, c_2)$</td>
<td>08 $\text{return } m$</td>
</tr>
</tbody>
</table>

**Listing 14:** Encryption and decryption function of the pre-shared-key APKE scheme $\text{pskAPKE} \{ \text{AKEM}, \text{KS}, \text{AEAD} \}$, built from AKEM, KS, and AEAD.

<table>
<thead>
<tr>
<th>$\text{pskAEnc}(sk, pk, psk, m, aad, info)$</th>
<th>$\text{pskADec}(pk, sk, (c_1, c_2), aad, info)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 $(c_1, K) \leftarrow \text{AuthEncap}(sk, pk)$</td>
<td>05 $K \leftarrow \text{AuthDecap}(pk, sk, c_1)$</td>
</tr>
<tr>
<td>02 $(k, \text{nonce}) \leftarrow \text{KS}(K, psk, info)$</td>
<td>06 $(k, \text{nonce}) \leftarrow \text{KS}(K, psk, info)$</td>
</tr>
<tr>
<td>03 $c_2 \leftarrow \text{AEAD.Enc}(k, m, aad, nonce)$</td>
<td>07 $m \leftarrow \text{AEAD.Dec}(k, c_2, aad, nonce)$</td>
</tr>
<tr>
<td>04 $\text{return } (c_1, c_2)$</td>
<td>08 $\text{return } m$</td>
</tr>
</tbody>
</table>

4.2 Security of $\text{pskPKE}$ and $\text{pskAPKE}$

The following Theorems 1–4 state privacy and authenticity of our constructions $\text{pskPKE} \{ \text{KEM}, \text{KS}, \text{AEAD} \}$ and $\text{pskAPKE} \{ \text{AKEM}, \text{KS}, \text{AEAD} \}$.

**Theorem 1** ($\text{KEM CCA} + \text{KS}_1 \text{ PRF} + \text{KS}_2 \text{ PRF} + \text{AEAD CCA} \Rightarrow \text{pskPKE CCA}$).

For any $(n, q_e, q_d, q_c, r_{pk}, r_{psk})$-CCA adversary $A$ against $\text{pskPKE} \{ \text{KEM}, \text{KS}, \text{AEAD} \}$, there exists an $(n, q_d, q_e)$-CCA adversary $B$ against $\text{KEM}$, a $(q_e, q_d)$-PRF adversary $C_1$ against $\text{KS}_1$, a $(q_e + q_d + q_c, q_e + q_d + q_c)$-PRF adversary $C_2$ against $\text{KS}_2$, and a $(q_e + q_c, q_d)$-CCA adversary $D$ against $\text{AEAD}$ such that

$$\text{Adv}_{A, \text{pskPKE}(\text{KEM}, \text{KS}, \text{AEAD})} \leq \text{Adv}_{B, \text{KEM}} + \text{Adv}_{C_1, \text{KS}_1} + \text{Adv}_{C_2, \text{KS}_2} + \text{Adv}_{D, \text{AEAD}} + q_e^2 + q_d^2 + q_c q_e.$$ 

**Proof (Sketch).** To prove CCA security for the $\text{pskPKE}$, we use CCA security of the underlying $\text{KEM}$ to replace the $\text{KEM}$ keys with uniformly random values. Together with a uniformly random $\text{psk}$, they can be used as PRF keys. Depending on the challenge query and which keys were corrupted, we can use either the PRF property when keyed on the
first (KS₁) or the second input (KS₂). For the second input we also need the KEM key to have enough entropy to avoid collisions. This yields random symmetric keys and the theorem follows by the CCA of AEAD. The full proof can be found in Appendix B.1. □

Theorem 2 (AKEM Insider-CCA + KS₁ PRF + KS₂ PRF + AEAD CCA ⇒ pskAPKE Insider-CCA). For any \((n, q_e, q_d, r_{pk}, r_{sk}, r_{psk})\)-Insider-CCA adversary \(A\) against the scheme \(\text{pskAPKE[AKEM, KS, AEAD]}\), there exists an \((n, q_e, q_d, r_{sk})\)-Insider-CCA adversary \(B\) against AKEM, a \((q_e, q_d + q_e)\)-PRF adversary \(C_1\) against KS₁, a \((q_e + q_d + q_e + q_d + q_e)\)-PRF adversary \(C_2\) against KS₂, and a \((q_e + q_d, q_d)\)-CCA adversary \(D\) against AEAD such that

\[
\text{Adv}(n, q_e, q_d, r_{pk}, r_{sk}, r_{psk}) & \text{-Insider-CCA}_A \\
\leq \text{Adv}(n, q_e, q_d, r_{sk}) & \text{-Insider-CCA}_B \\
+ & \text{Adv}(q_e, q_d + q_e) & \text{-PRF}_{C_1, KS_1} \\
+ & \text{Adv}(q_e + q_d + q_e + q_d + q_e) & \text{-PRF}_{C_2, KS_2} \\
+ & \text{Adv}(q_e + q_d) & \text{-CCA}_{D, AEAD} \\
+ & q_e^2 + q_d^2 + q_e q_d \frac{2^n}{2^\eta}.
\]

Proof (Sketch). Since we want to achieve Insider-CCA security for the pskAPKE, we have to simulate the RepSK oracle which can be done by reducing to an Insider-CCA secure AKEM. The remaining part of the proof is essentially the same as for Theorem 1. The full proof can be found in Appendix B.2. □

Theorem 3 (KEM CCA + KS₁ PRF + KS₂ PRF + AEAD INT-CTX ⇒ pskPKE Auth). For any \((n, q_e, q_d, r_{pk}, r_{psk})\)-Auth adversary \(A\) against the scheme \(\text{pskPKE[KEM, KS, AEAD]}\), there exists a \((n, q_d, q_e)\)-CCA adversary \(B\) against KEM, a \((q_e, q_e)\)-PRF adversary \(C_1\) against KS₁, a \((q_d, q_d)\)-PRF adversary \(C_2\) against KS₂, and a \((2q_e + q_d + 1, q_d + 1)\)-INT-CTX adversary \(D\) against AEAD such that

\[
\text{Adv}(n, q_e, q_d, r_{pk}, r_{psk}) & \text{-Auth}_A \\
\leq \text{Adv}(n, q_d, q_e) & \text{-CCA}_B \\
+ & \text{Adv}(q_e, q_e) & \text{-PRF}_{C_1, KS_1} \\
+ & \text{Adv}(q_d, q_d) & \text{-PRF}_{C_2, KS_2} \\
+ & \text{Adv}(2q_e + q_d + 1, q_d + 1) & \text{-INT-CTX}_{D, AEAD} \\
+ & q_e (q_e + q_d - 1) \frac{|K|}{2^n}.
\]

Proof (Sketch). We use the CCA security of KEM to ensure that the key \(K\) fed into KS₂ is uniform random such that, with high probability, there are no key collisions. Together with the pre-shared key, at least one of the inputs is uniformly random and the PRF property of KS yields a random output. Then, this output can be used for the AEAD such that decryption queries (with respect to an honest psk) can be rejected. The full proof can be found in Appendix B.3. □

Theorem 4 (AKEM Outsider-CCA + AKEM Outsider-Auth + KS₁ PRF + KS₂ PRF + AEAD INT-CTX ⇒ pskAPKE Outsider-Auth). For any \((n, q_e, q_d, r_{pk}, r_{psk})\)-Outsider-Auth adversary \(A\) against the scheme \(\text{pskAPKE[AKEM, KS, AEAD]}\), there exists an \((n, 0, q_e)\)-CCA adversary \(B\) against AKEM, a \((q_e + q_d, q_e + q_d)\)-PRF adversary \(C_1\) against KS₁, a \((q_e + q_d, q_e + q_d)\)-PRF adversary \(C_2\) against KS₂, and a \((2q_e + q_d + 1, q_d + 1)\)-INT-CTX adversary \(D\)
Proof (Sketch). To achieve Outsider-Auth security for pskAPKE, we need to first use Outsider-CCA security and Outsider-Auth security of AKEM to replace the KEM secret in encryption and decryption by random values. Together with the pre-shared keys they can be used as inputs to $KS$ where, depending on the query, either the KEM shared secret or the $psk$ act as the PRF key. Next, the PRF output can be used to construct an adversary against INT-CTXT security of AEAD. The full proof can be found in Section 4.4. □

4.3 The Security of HPKE’s PSK Modes

The HPKE standard’s specification of the HPKE$_{PSK}$ mode corresponds to the construction $pskPKE[KEM, KS, AEAD]$ (Listing 13), and the one of the HPKE$_{AuthPSK}$ mode to the construction $pskAPKE[AKEM, KS, AEAD]$ (Listing 14) with one exception. The HPKE standard explicitly defines an identifier for each $psk$, the $psk_{-id}$, and the actual key schedule function takes it as an additional parameter alongside the KEM key $K$, the $psk$, and the info bitstring that we consider in our model. For simplicity, we abstract away the $psk_{-id}$ and consider it to be encoded as part of $info$, as both are simply hashed into the context of the key derivation. HPKE uses the following specific components:

- KEM is the standard Diffie-Hellman $DH$-KEM which fulfills CCA security assuming the Gap Diffie-Hellman assumption.
- AKEM is the Diffie-Hellman $DH$-AKEM from [1] which is proved Insider-CCA and Outsider-Auth-secure assuming the Gap Diffie-Hellman assumption.
- The key schedule $KS$ is constructed via the functions Extract and Expand both instantiated with HMAC [1, Section 6.2]. If we assume HMAC to be a PRF when keyed on either of the inputs, the assumptions for $KS$ hold as well.
- AEAD is instantiated using AES-GCM or ChaCha20-Poly1305, which are shown to fulfill IND-CPA and INT-CTXT security [7] and thus also IND-CCA security.

Thus, applying the composition theorems from the last section, we achieve CCA and Auth security for HPKE$_{PSK}$, and Insider-CCA and Outsider-Auth security for HPKE$_{AuthPSK}$.

4.4 Proof of Theorem 4

Proof. We describe several games depicted in Listing 15.

Game $G_0$. This is the $(n, q_e, q_d, r_{pk}, r_{psk})$-Outsider-Auth game for pskAPKE[AKEM, KS, AEAD], thus we have

$$\Pr[G_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, r_{pk}, r_{psk}) \text{-Outsider-Auth}(\mathcal{A}) \Rightarrow 1].$$
Listing 15: Games $G_0$ – $G_8$ for the proof of Theorem 4.

Game $G_1$. We insert a set $\Lambda$ to log the outputs of $KS$ to use the stored outputs if $KS$ is queried on the same parameters again. This is only done for encryption oracle queries for which the receiver’s key was not replaced, i.e. $j \notin \Gamma_{pk}$ (checked in Line 13), as well as for decryption queries for which not both the sender’s key and the $psk$ were replaced, i.e. for queries on indices $i \notin \Gamma_{pk} \lor (i, j) \notin \Gamma_{psk}$ (checked in Line 39). Since the change is only conceptual, we have

$$\Pr[G_0 \Rightarrow 1] = \Pr[G_1 \Rightarrow 1].$$

Game $G_3$. If the corresponding receiver key has not been corrupted via $RepPK$, i.e. $j \notin \Gamma_{pk}$, we replace the KEM secret in oracle $Enc$ by a uniformly random value (Line 14) and store the output to return consistent queries to oracle $Dec$. The difference is the advantage of a CCA adversary $B_1$ against $AKEM$:

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \text{Adv}_{B_1, AKEM}^{(n,q_0,q_0,q_2,\text{CCA})}.$$
Game $G_3$. In $G_3$, the game aborts in the encryption oracle if the corresponding receiver’s key was not replaced and there already exists an entry in $\Lambda$ with the queried parameters (Line 18). The difference is negligible in the size of the key space of KEM. Since $K$ was randomly chosen in the previous game, the probability of having such an entry is at most $|\Lambda|/|K|$. Note that $\Lambda$ is filled with another element at most once per ENC/DEC query. This yields the following advantage:

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq q_e(q_e + q_d - 1)/|K|.$$ 

Game $G_4$. If the receiver’s key was not replaced and we do not abort, we replace the output of $K_{S_1}$ in the encryption oracle with uniformly random values of the respective domain (Line 19). The game difference is the advantage of a PRF adversary $C_1$ against $K_{S_1}$:

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq \text{Adv}_{^{\text{PRF}}_{C_1,K_{S_1}}}^{(q_e,q_d)}.$$ 

The changes in Game $G_1$ ensure consistent outputs of $K_{S_1}$, i.e. queries on ENC or DEC with the same parameters lead to the same output of $K_{S_1}$ (or the game aborts in the for pskEnc). Hence, games $G_3$ and $G_4$ can only be distinguished by distinguishing the real output of $K_{S_1}$ from a uniformly random one. This can be turned into an adversary against PRF security of $K_{S_1}$, i.e. keyed on the first input. Note that $K$ is chosen uniformly at random due to the changes in Game $G_2$. There are at most $q_e$ different instances for the PRF and at most the same number of queries.

Game $G_6$. The game is modified by choosing uniformly random values instead of the real output of $K_{S_1}$ in the same case as for the previous game (Line 32). This can be turned into an adversary against PRF security of $K_{S_1}$, i.e. keyed in the first input. There are at most $q_d$ different instances and at most $q_d$ different queries resulting in

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq \text{Adv}_{^{\text{PRF}}_{C_2,K_{S_1}}}^{(q_d,q_d)}.$$ 

Game $G_7$. We modify the game by replacing the output of $K_{S_1}$ by uniformly random values similar to the last game modification but in the following case while querying the decryption oracle: not both the sender’s key and the $psk$ were corrupted ($i \not\in I_{pk} \lor (i, j) \not\in I_{psk}$, Line 39), there is no corresponding element in $\Lambda$ (Line 42), and the sender’s key was corrupted, i.e. $i \in I_{pk}$ (Line 42). This can be turned into a PRF adversary $C_2$ against $K_{S_2}$, i.e. keyed in the second input:

$$|\Pr[G_7 \Rightarrow 1] - \Pr[G_8 \Rightarrow 1]| \leq \text{Adv}_{^{\text{PRF}}_{C_2,K_{S_2}}}^{(q_d,q_d)}.$$
The two conditions \( i \notin \Gamma_{pk} \lor (i, j) \notin \Gamma_{pak} \) and \( i \in \Gamma_{pk} \) imply that the \( \text{psk} \) was not replaced and was therefore chosen uniformly at random in the beginning of the game. Thus, the two games can be simulated by adversary \( C_2 \) via their own evaluation oracle. We have at most \( q_d \) different indices for the \( \text{PRF} \) game and at most the same number of queries.

**Game \( G_8 \).** In this game, we replace the actual decryption in an honest decryption oracle query with \( \bot \) (Line 45). Distinguishing the game difference can be turned into an \( \text{INT-CTXT} \) adversary \( D_1 \) against AEAD:

\[
\Pr[G_7 \Rightarrow 1] - \Pr[G_8 \Rightarrow 1] \leq \text{Adv}^{(q_e + q_d, q_a)}_{D_1, \text{AEAD}}
\]

Adversary \( D_1 \) is formally constructed in Listing 16. Note that \( k \) and \( \text{nonce} \) are uniformly random such that the adversary can use their own decryption oracle either on a new index or on a previous index in case the same parameters were queried before and the element is in \( A \). Further, encryption queries can also be simulated in each case. If there is an encryption query with a corrupted receiver’s key, the adversary can compute the encryption on their own. Otherwise, they can use their own encryption oracle. The abort in cases of a parameter set being queried before (Line 18) prevents the need of querying the encryption oracle twice which is not possible for the \( \text{INT-CTXT} \) game for an AEAD. There are at most \( q_e + q_d \) different keys and adversary \( D_1 \) makes at most \( q_d \) queries to their decryption oracle \( \text{Dec}_{\text{AEAD}} \).

**Listing 16:** Adversary \( D_1 \) against \( \text{INT-CTXT} \) security for AEAD having access to oracles \( \text{Enc}_{\text{AEAD}} \) and \( \text{Dec}_{\text{AEAD}} \).
Reduction to Game $G_8$. Winning Game $G_8$ can be reduced to an INT-CTX adversary $D_2$ against APKE:

$$\Pr[G_8 \Rightarrow 1] \leq \text{Adv}^{(q_c+1,1)}_{D_2,\text{AEAD}}(\cdot).$$

The adversary can simulate the decryption oracle since in cases $i \notin \Gamma_{\text{psk}}$ or $(i,j) \notin \Gamma_{\text{psk}}$, they can output $\perp$ (or the original encryption if it was produced during the experiment). In cases $(i,j) \in \Gamma_{\text{psk}}$, they can compute the output by their own. For the encryption oracle, the adversary can use their own encryption oracle or compute the output on their own similar to the adversary from the last game hop. The output of the adversary against game $G_8$ can then be used to issue a decryption query in the INT-CTX experiment on either a new key or a previous key if the output parameters $i^*, j^*, c^*_i, \text{info}^*$ match with that key. Matching parameters can be identified by computing $K^* \leftarrow \text{Decaps}(sk_{j^*}, c^*_i)$ and comparing $(i^*, j^*, K^*, psk_{i,j}^*, \text{info}^*)$ to set $A$ similar to Line 16 or Line 39 in Listing 16. If the adversary against $G_8$ wins, the adversary against the INT-CTX experiment has a valid ciphertext which does not decrypt to $\perp$ due to the winning condition of $G_8$. That means they can distinguish between the real or random case since the result of the decryption query must be unequal to $\perp$ in the real case.

Putting everything together, we obtain the stated bound.  

\[\tag{4}\]

5 Hybrid Post-Quantum APKE

We want to build an HPKE scheme which is secure against classical as well as quantum adversaries. To not rely solely on relatively new post-quantum primitives, a common way is to use combiners which combine well studied classical primitives and post-quantum primitives at the same time. This hybrid approach allows for security against future quantum adversaries but is still secure in a classical setting if current post-quantum primitives are broken. To this end, we use the pre-shared key mode of HPKE to build a combiner from which we can instantiate a hybrid post-quantum construction.

Let $\text{pskAPKE}[\text{AKEM}_1, \text{KS}, \text{AEAD}]$ be a pre-shared key PKE based on an authenticated KEM $\text{AKEM}_1 = (\text{Gen}_1, \text{AuthEncap}_1, \text{AuthDecap}_1)$, a two-keyed function $\text{KS}$, and an authenticated encryption with associated data AEAD as in Listing 14. Further, let $\text{AKEM}_2 = (\text{Gen}_2, \text{AuthEncap}_2, \text{AuthDecap}_2)$ be a second authenticated KEM. From these components, we can construct an APKE using the shared secret of the second KEM as the pre-shared key of the $\text{pskAPKE}$. We remark that the same construction also works for non-authenticated primitives, i.e., we can construct a PKE $\text{PKE}[\text{KEM}_1, \text{KEM}_2, \text{KS}, \text{AEAD}]$ built from $\text{pskPKE}[\text{KEM}_1, \text{KS}, \text{AEAD}]$ and a $\text{KEM}_2$.

The following two theorems state that the APKE is secure (in the sense of Insider-CCA and Outsider-Auth) if at least one of the underlying AKEMs, $\text{AKEM}_1$ or $\text{AKEM}_2$, is secure.

**Theorem 5.** Let $\text{AKEM}_1$ and $\text{AKEM}_2$ be two AKEMs, $\text{KS}$ a two-keyed function, and AEAD an AEAD. If $\text{KS}$ is a PRF in both keys, AEAD is IND-CCA secure, and $\text{AKEM}_1$ or $\text{AKEM}_2$ is CCA secure, then the construction in Listing 17 is a CCA secure APKE. In particular, for any $(n, q_c, q_d, q_e)$-Insider-CCA adversary $A$ against $\text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{KS}, \text{AEAD}]$ there exists a $(n+1, q_c, q_d, q_e)$-Insider-CCA adversary $B_1$ against $\text{AKEM}_1$, a $(n, q_c, q_d, q_e)$-Insider-CCA adversary $B_2$ against $\text{AKEM}_2$, a $(q_c, q_d + q_c)$-PRF adversary $C_1$ against $\text{KS}_1$, a $(q_c, q_d + q_c)$-PRF adversary $C_2$ against $\text{KS}_2$, and a $(q_c, q_d)$-CCA adversary $D$ against AEAD.
such that

\[
\text{Adv}^{(n,q_e,q_d,q_t)}_{\text{Insider-CCA}} \leq \\
\min \{ \text{Adv}^{(n+1,q_e,q_d,q_t)}_{B_1,\text{AKEM}_1} + \text{Adv}^{(q_e,q_d+q_c)}_{C_1,\text{KS}_1}, \text{Adv}^{(n,q_e,q_d,q_t)}_{B_2,\text{AKEM}_2} + \text{Adv}^{(q_e,q_d+q_c)}_{C_2,\text{KS}_2} \} + \text{Adv}^{(q_e,q_d)}_{\text{PRF},\text{AEAD}}
\]

Proof (Sketch). The first part of the proof is very similar to Theorem 2 except that the queries to KS2 can be saved. The second part transforms the KEM secret of AKEM2 into a uniformly random value using its Insider-CCA security which can then be used as input to KS2 as a regular psk. These outputs are uniformly random values and the remaining transformations are as for the first part. The full proof can be found in Appendix C.1. □

**Theorem 6.** Let AKEM1 and AKEM2 be two AKEMs, KS a two-keyed function, and AEAD an AEAD. If KS is a PRF in both keys, AEAD is INT-CTX and IND-CPA secure, and AKEM1 or AKEM2 is Outsider-Auth secure, then the construction in Listing 17 is a Outsider-Auth secure APKE. In particular, for any \((n,q_e,q_d)\)-Outsider-Auth adversary \(A\) against APKE[AKEMK1,AKEM2,KS,AEAD], there exists a \((n+1,0,q_e,q_d)\)-Outsider-CCA adversary \(B_1\) against AKEM1, a \((n+1,q_e,q_d)\)-Outsider-Auth adversary \(B_2\) against AKEM1, a \((n,0,q_e,q_d)\)-Outsider-CCA adversary \(B'_1\) against AKEM2, a \((n,q_e,q_d)\)-Outsider-Auth adversary \(B'_2\) against AKEM2, a \((q_e+q_d,q_e+q_d)\)-PRF adversary \(C_1\) against KS1, a \((q_e,q_d)\)-PRF adversary \(C_2\) against KS2, a \((2q_e+q_d+1,q_d+1)\)-INT-CTX adversary \(D\) against AEAD such that

\[
\text{Adv}^{(n,q_e,q_d)}_{\text{Outsider-Auth}} \leq \\
\min \{ \text{Adv}^{(n+1,q_e,q_d,q_t)}_{B_1,\text{AKEM}_1} + \text{Adv}^{(n+1,q_e,q_d)}_{B_2,\text{AKEM}_1}, \text{Adv}^{(n,q_e,q_d)}_{B'_1,\text{AKEM}_2} + \text{Adv}^{(n,q_e,q_d)}_{B'_2,\text{AKEM}_2}, \text{Adv}^{(q_e+q_d)}_{C_1,\text{KS}_1}, \text{Adv}^{(q_e,q_d)}_{C_2,\text{KS}_2} \} + \text{Adv}^{(2q_e+q_d+1)\text{-INT-CTX}}_{D,\text{AEAD}} + \frac{q_e(q_e+q_d-1)}{|K|}.
\]
Proof (Sketch). The first part of the proof is very similar to Theorem 4 except that the queries to $KS_2$ can be saved. The second part transforms the KEM secret of $AKEM_2$ into a uniformly random value using its Insider-CCA and Outsider-Auth security which can then be used as input to $KS_2$ as a regular $psk$ in encryption and decryption oracle. These outputs are uniformly random values and the remaining transformations are as for the first part. The full proof can be found in Appendix C.2.

Post-Quantum Instantiation. Consequently, one can combine $HPKE_{AuthPSK}$ with a post-quantum secure $AKEM$ to obtain an APKE scheme with hybrid security. Analogously, one can combine $HPKE_{PSK}$ with a post-quantum secure $KEM$ (such as Kyber) to obtain a PKE with hybrid security. In the next section, we discuss how to construct post-quantum secure $AKEM$ schemes.

6 Post-Quantum AKEM Constructions

6.1 KEM-then-Sign-then-Hash

A well-known approach for constructing a post-quantum AKEM is to combine a post-quantum KEM with a post-quantum signature [10]. This could obviously applied to the classical setting as well but with much worse performance than the NIKE-based construction of HPKE which achieves authentication almost for free.

Our new construction extends the (insecure) Encrypt-then-Sign (EtS) paradigm to Encrypt-then-Sign-then-Hash (EtStH). Let $KEM = (KEM.Gen, Encaps, Decaps)$ be a KEM and $SIG = (Gen, Sign, Vfy)$ be a signature scheme. We construct $AKEM^{EtStH}[KEM, SIG, H]$ as shown in Listing 18 for $H$ a keyed function. We define the output of $H$ to be $\perp$ if one of the inputs is $\perp$. The key generation outputs a public key tuple and a private key tuple. The first component of both tuples is the receiver’s public/private key and the second component is the sender’s public/private key.

Listing 18: $AKEM^{EtStH}[KEM, SIG, H]$ from a KEM $KEM = (KEM.Gen, Encaps, Decaps)$, a signature scheme $SIG = (SIG.Gen, Sign, Vfy)$, and a random oracle $H$.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>$Gen((sk, pk) \leftarrow KEM.Gen)$</td>
</tr>
<tr>
<td>02</td>
<td>$ Sigk, vk \leftarrow SIG.Gen$</td>
</tr>
<tr>
<td>03</td>
<td>return $((sk, sigk), (pk, vk))$</td>
</tr>
<tr>
<td>04</td>
<td>$c, K' \leftarrow Encaps(pk_2)$</td>
</tr>
<tr>
<td>05</td>
<td>$\sigma \leftarrow Sign(sigk_1, c</td>
</tr>
<tr>
<td>06</td>
<td>$K \leftarrow H(K', \sigma</td>
</tr>
<tr>
<td>07</td>
<td>return $((c, \sigma), K)$</td>
</tr>
<tr>
<td>08</td>
<td>if $Vfy(vk_1, c</td>
</tr>
<tr>
<td>09</td>
<td>$K \leftarrow \perp$</td>
</tr>
<tr>
<td>10</td>
<td>else</td>
</tr>
<tr>
<td>11</td>
<td>$K' \leftarrow Decaps(sk_2, c)$</td>
</tr>
<tr>
<td>12</td>
<td>$K \leftarrow H(K', \sigma</td>
</tr>
<tr>
<td>13</td>
<td>return $K$</td>
</tr>
</tbody>
</table>

Theorem 7 (KEM CCA + H PRF $\Rightarrow$ AKEM Insider-CCA). If $KEM$ is a CCA secure key encapsulation mechanism and $H$ is a PRF, then $AKEM^{EtStH}[KEM, SIG, H]$ is an Insider-CCA
secure AKEM. In particular, for every \((n, q_c, q_d, q_e, r_\text{sk})\)-Insider-CCA adversary \(A\) against AKEM\textsuperscript{\text{EstH}}[\text{KEM, SIG, H}]\) there exists a \((n, q_d, q_e)\)-CCA adversary \(B\) against KEM and a \((q_e, q_d + q_e)\)-PRF adversary \(C\) against H such that

\[
\text{Adv}_{A,\text{AKEM}\text{EstH}[\text{KEM, SIG, H}]}^{(n, q_c, q_d, q_e, r_\text{sk})\text{-Insider-CCA}} \leq \text{Adv}_{B,\text{KEM}}^{(n, q_d, q_e)\text{-CCA}} + \text{Adv}_{C,\text{H}}^{(q_e, q_d + q_e)\text{-PRF}}.
\]

Proof (Sketch). We use the CCA security of KEM to make the KEM keys random, such that the key to H is uniformly random. Using the PRF property of H gives a uniformly random value for the final key. The full proof can be found in Appendix D.1. □

Theorem 8 (SIG SUF-CMA ⇒ AKEM Outsider-Auth). If SIG is an SUF-CMA secure signature scheme, then AKEM\textsuperscript{\text{EstH}}[\text{KEM, SIG, H}] is an Outsider-Auth secure AKEM. In particular, for every \((n, q_e)\)-Outsider-Auth adversary \(A\) there exists a \((n, q_e)\)-SUF-CMA adversary \(B\) against SIG such that

\[
\text{Adv}_{A,\text{AKEM}\text{EstH}[\text{KEM, SIG, H}]}^{\text{Outsider-Auth}} \leq \text{Adv}_{B,\text{SIG}}^{(n, q_e)\text{-SUF-CMA}}.
\]

Proof (Sketch). Queries to the decapsulation oracle containing invalid signatures cannot be distinguished by an adversary due to the definition of the scheme. Valid queries can be used against the SUF-CMA security of SIG. The full proof can be found in Appendix D.2. □

6.2 AKEM from NIKE

We can build an AKEM from a NIKE. Let NIKE = (Setup, NIKE.KeyGen, NIKE.SharedKey) be a NIKE and H a 2-keyed function, then we can construct an AKEM AKEM\textsuperscript{NIKE}[NIKE, H] as defined in Listing 19. We define the output of H to be \(\perp\) if one of the inputs is \(\perp\). By \(H_1\), we denote function H keyed in the first component and by \(H_2\) function H keyed in the second component. In contrast to an earlier version of the paper, hash function H takes both the users’ public keys and the ciphertext \(pk^*\) as additional input. This change prevents two attacks possible in the previous version. If the sender and receiver keys are not hashed and adversary can query the decapsulation oracle on the output of an encapsulation with switched roles of sender and receiver. If the ciphertext is not hashed, an adversary could find a collision in the remaining inputs to the hash function and exploit this to distinguish the output of the decapsulation oracle.

Theorem 9 (NIKE Active + H\textsubscript{2} PRF ⇒ AKEM Insider-CCA). Let NIKE be a NIKE and H a 2-keyed function. If NIKE is Active secure and H\textsubscript{2} a PRF, then AKEM\textsuperscript{NIKE}[NIKE, H] is Insider-CCA secure. In particular for any adversary \(A\) against \((n, q_c, q_d, q_e, r_\text{sk})\)-Insider-CCA security of AKEM\textsuperscript{NIKE}[NIKE, H] there exists an \((n + q_c, q_e + 2q_d, 0, q_e + q_d + q_c, q_e + 2q_d, q_e)\)-Active adversary against NIKE and a \((q_e, q_e)\)-PRF adversary \(C\) against H\textsubscript{2} such that

\[
\text{Adv}_{A,\text{AKEM}^{\text{NIKE}}[\text{NIKE, H}]}^{(n, q_c, q_d, q_e, r_\text{sk})\text{-Insider-CCA}} \leq \text{Adv}_{B,\text{NIKE}}^{(n + q_c, q_e + 2q_d, 0, q_e + q_d + q_c, q_e + 2q_d, q_e)\text{-Active}} + \text{Adv}_{C,\text{H}_2}^{(q_e, q_e)\text{-PRF}}.
\]

Proof (Sketch). Assuming an active secure NIKE, the second shared key, \(K_2\) is indistinguishable from random. We show that by constructing an adversary against an Active secure NIKE using an Insider-CCA adversary against AKEM\textsuperscript{NIKE}[NIKE, H] by simulating the
corruptions from RepSK by registering corrupt users in the NIKE game. Then, every other query can be answered by registering a new key (if the query was made with a chosen public key) or computed by the simulator themselves. The test query of the adversary against NIKE is then directly embedded int the challenge query of the Insider-CCA game. With $H_2$ being a PRF, we can further show that the resulting key is also uniformly random. The full proof can be found in Appendix D.3.

**Theorem 10 (NIKE Active + $H_1$ PRF $\Rightarrow$ AKEM Outsider-Auth).** Let $\text{NIKE} = (\text{Setup, NIKE.KeyGen, NIKE.SharedKey})$ be a NIKE and $H$ a 2-keyed function. If NIKE is Active secure and $H_1$ a PRF, then $\text{AKEM}^{\text{NIKE}}[\text{NIKE}, H]$ is Outsider-Auth secure. In particular, for every $(n, q_e, q_d)$-Outsider-Auth adversary against $\text{AKEM}^{\text{NIKE}}[\text{NIKE}, H]$ there exists an $(n, q_e + 2q_d, 0, q_e, 2q_e + 2q_d, q_d)$-Active adversary against NIKE and a $(q_d, q_d)$-PRF adversary $C$ against $H_1$ such that

$$\text{Adv}_{\text{A,AKEM}^{\text{NIKE}}[\text{NIKE}, H]}^{\text{NIKE,Outsider-Auth}} \leq \text{Adv}_{\text{B,NIKE}}^{(n, \mu_q + \mu_d, 0, 0, q_e, \mu_q, q_e + \mu_d)-\text{Active}} + \text{Adv}_{C,H_1}^{(q_d, q_d)-\text{PRF}} + q_e(q_e + q_d)P_{\text{NIKE}}$$

**Proof (Sketch).** The structure is similar to the proof of Theorem 9 except that the test query is embedded in both encapsulation and the decapsulation oracle and that it is only embedded for queries with honest public keys. Further we need the entropy of the NIKE public key to avoid attacks using collisions in the hash function. The full proof can be found in Appendix D.4.

**Acknowledgements.** The authors thank the anonymous reviewers to point out an error in our NIKE construction and an error in one of our proofs. They also thank Doreen Riepel for very helpful feedback and discussions. Jonas Janneck was supported by the European Union (ERC AdG REWORC - 101054911). Eike Kiltz was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy – EXC 2092 CASA - 390781972, and by the European Union (ERC AdG REWORC - 101054911).
References


A Omitted Security Definition

A.1 Active Security NIKE

The following definition also known as m-CKS-heavy is taken from [13].

Definition 12 (Active Security). Let \( \text{NIKE} = (\text{Setup}, \text{NIKE}, \text{SharedKey}) \) be a NIKE. We define active security of \( \text{NIKE} \) (Active) via game \((q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)\)-Active in Listing 20. The advantage of an adversary \( A \) is

\[
\text{Adv}_{A,\text{NIKE}}^{(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)\text{-Active}} = \left| \Pr[(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)\text{-Active}(A) \Rightarrow 1] - \frac{1}{2} \right|.
\]

B Proofs for the pskPKE and pskAPKE constructions

B.1 Proof of Theorem 1

Proof. We construct a sequence of games as shown in Listing 21.

Game \( G_0 \). \( G_0 \) is the \((n, q_e, q_d, q_c, r_{pk}, r_{psk})\)-CCA game for pskPKE[KEM, KS, AEAD] (cf. Section 3.2) executed with an adversary \( A \), i.e.,

\[
\Pr[G_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, q_c, r_{pk}, r_{psk})\text{-CCA}(A) \Rightarrow 1].
\]
Listing 20: Game \((q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)\)-Active for NIKE. Adversary \(\mathcal{A}\) makes at most \(q_H\) queries to \(\text{REGHonest}\), at most \(q_C\) queries to \(\text{REGCORRUPT}\), at most \(q_E\) queries to \(\text{EXTRACT}\), at most \(q_{HR}\) queries to \(\text{HONESTREV}\), at most \(q_{CR}\) queries to \(\text{CORRUPTREV}\), and at most \(q_T\) queries to \(\text{TEST}\). The total number of users is denoted by \(n = q_H + q_C\).

\[
\begin{array}{|c|}
\hline
(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)\text{-Active} \\
01 & E, T, R \leftarrow \emptyset \\
02 & b \leftarrow \{0, 1\} \\
03 & \mathcal{A}^{\text{REGHonest, REGCORRUPT, EXTRACT, HONESTREV, CORRUPTREV, TEST}} \\
04 & \text{return } [b = b' \land \exists (i, j) \in \mathcal{R} : (i, j, \cdot) \in \mathcal{T} \\
& \land \exists i \in E : (i, \cdot, \cdot) \in \mathcal{T} \land \exists j \in E : (\cdot, j, \cdot) \in \mathcal{T}] \\
\text{Oracle } \text{REGHonest}(i \in [n]) & \\
05 & \text{if } pk_i \neq \bot \land sk_i = \bot \text{ return } \bot \\
06 & \text{return } \bot \\
07 & (sk_i, pk_i) \leftarrow \text{NIKE.KeyGen} \\
08 & \text{return } pk_i \\
\text{Oracle } \text{REGCORRUPT}(i \in [n], pk) & \\
09 & \text{if } sk_i \neq \bot \text{ return } \bot \\
10 & \text{return } \bot \\
11 & pk_i \leftarrow pk \\
12 & sk_i = \bot \\
\text{Oracle } \text{EXTRACT}(i \in [n], pk) & \\
13 & E \leftarrow E \cup \{i\} \\
14 & \text{return } sk_i \\
\end{array}
\]

Oracle \text{HONESTREV}(i \in [n], j \in [n])

\[
\begin{array}{c}
15 & \text{if } sk_i = \bot \lor sk_j = \bot \\
16 & \text{return } \bot \\
17 & \mathcal{R} \leftarrow \mathcal{R} \cup \{(i, j), (j, i)\} \\
18 & \text{return } \text{NIKE.SharedKey}(sk_i, pk_j) \\
\end{array}
\]

Oracle \text{CORRUPTREV}(i \in [n], j \in [n])

\[
\begin{array}{c}
19 & \text{if } pk_i = \bot \lor pk_j = \bot \\
20 & \text{return } \bot \\
21 & \mathcal{R} \leftarrow \mathcal{R} \cup \{(i, j), (j, i)\} \\
22 & \text{if } sk_i \neq \bot \land sk_j = \bot \\
23 & \text{return } \text{NIKE.SharedKey}(sk_i, pk_j) \\
24 & \text{if } sk_i = \bot \land sk_j \neq \bot \\
25 & \text{return } \text{NIKE.SharedKey}(sk_j, pk_i) \\
26 & \text{return } \bot \\
\end{array}
\]

Oracle \text{TEST}(i \in [n], j \in [n])

\[
\begin{array}{c}
27 & \text{if } i = j \lor sk_i = \bot \lor sk_j = \bot \\
28 & \text{return } \bot \\
29 & \text{if } \exists K : (i, j, K) \in \mathcal{T} \\
30 & \text{return } K \\
31 & \text{if } b = 0 \\
32 & K \leftarrow \text{NIKE.SharedKey}(sk_i, pk_j) \\
33 & \text{else} \\
34 & K \leftarrow \text{SHK} \\
35 & \mathcal{T} \leftarrow \mathcal{T} \cup \{(i, j, K)\} \\
36 & \text{return } K \\
\end{array}
\]

Game \(G_1\). In this game, we make the following changes. If the challenge oracle \text{CHALL} is queried on a receiver index \(j\) which was not replaced (i.e., \(j \notin I_{pk}\)) we replace the KEM key \(K\) by a uniformly random value from the key space \(K\). The result is then stored together with the receiver’s public key and the ciphertext in set \(\mathcal{E}'\). For consistency, we further change the decryption oracle \text{DEC} by replacing the KEM key \(K\) by the one from set \(\mathcal{E}'\) if there already exists such a key for the queried ciphertext \(c_1\) and the receiver’s public key \(pk_j\). Distinguishing these changes can be turned into an adversary \(\mathcal{B}\) against \((n, q_d, q_c)\)-CCA security of KEM as depicted in Listing 22.

\[
\Pr[G_0 = 1] - \Pr[G_1 = 1] \leq \text{Adv}_{\mathcal{B}}^{(n, q_d, q_c)\text{-CCA}}.
\]

Note that \(\mathcal{B}\) can simulate all oracles by their own or with the oracles provided by their own experiment. For the \text{REPLACE} oracle, the reduction has to store the indices but can abort with \(\bot\) for corresponding decryption oracle queries, i.e. decryption queries for which \(sk_j = \bot\) due to Line 23 in the game. The decryption oracle can be simulated by the decapsulation oracle provided by the KEM CCA game. If the challenge oracle is queried on non-replaced receiver keys, i.e., \(j \notin I_{pk}\) (Line 46 in the game), \(\mathcal{B}\) can use their own
If $B$ is in the real case of their $(n, q_d, q_a)$-CCA game (case $b = 0$ in Listing 1), they perfectly simulate $G_0$. If they are in the random case (case $b = 1$ in Listing 1), they perfectly simulate $G_1$. Any other encapsulation calls can be computed by the reduction itself.

**Game $G_2$.** In Game $G_2$, we change the challenge oracle by replacing the output of $KS$ by uniformly random values of the respective domain if the receiver’s key was not replaced, i.e. if $j \notin \Gamma_{pk}$ (Line 46). Input and output are also logged using set $A_K$ (Line 51) to return consistent decapsulation queries. To this end, the decapsulation oracle is modified such that for $KS$ it returns a previous output stored in $A_K$ (Line 30) or a uniformly random as well (Line 32). The difference can be turned into a PRF adversary $C_1$ against $KS_1$, i.e. $KS$ keyed in the first component:

$$\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1] \leq \text{Adv}^{(q_d, q_a, q_e)}_{C_1, KS_1}.$$
The difference can be upper bounded by using the key entropy of collide which includesability which leaves us with the probability of an abort. If the game aborts, the inputs to
Line 56). Otherwise, both oracles
before, i.e. a matching element in
Decapsulation operations, i.e. made from
Encapsulation operations, i.e. made from
KS
calls to
Enc
Game
G
constructed in Listing 23. Note that such an adversary needs at most
q
c
different PRF keys and at most
q
d +
q
e
queries to the PRF evaluation since they have to simulate KS
queries in the decapsulation and challenge oracle.

**Game G_2.** In G_2, the game aborts if there is a collision in the inputs to KS when called from oracles ENC or CHALL in case of a non-replaced pre-shared key, i.e. \((i, j) \notin \Gamma_{pak}\).
The difference can be upper bounded by using the key entropy of KEM. To log the calls to KS, we introduce sets \(A_{Enc}\) and \(A_{Dec}\), where \(A_{Enc}\) corresponds to calls after encapsulation operations, i.e. made from ENC and CHALL, and \(A_{Dec}\) corresponds to calls after decapsulation operations, i.e. made from DEC. We distinguish these cases because the game only aborts for collisions in ENC or CHALL (Line 14 and Line 54). If there is no such collision, i.e. no matching element in \(A_{Enc}\), but the same inputs were queried in \(A_{Dec}\) before, i.e. a matching element in \(A_{Dec}\), the stored outputs are used (Line 16 and Line 56). Otherwise, both oracles ENC and CHALL store inputs and outputs to KS in \(A_{Enc}\) and proceed (Line 20 and Line 60). The changes in the decryption oracle are only for consistency. If there was a matching query before, the stored input is used (Line 36) and otherwise it is stored in set \(A_{Dec}\) (Line 39).

Since KS is deterministic, using the stored values does not change the winning probability which leaves us with the probability of an abort. If the game aborts, the inputs to KS collide which includes K. However, K is the output of a KEM encapsulation, i.e. the probability of a collision in K is \(1/2^\eta\) for \(\eta\) being the key entropy of KEM. This leads to a probability of at most \(2\eta^{-1}(q_c+q_d)\) for the abort in ENC and at most \(2\eta^{-1}(q_c+q_d)\) for the abort

---

**Listing 22:** Adversary B against \((n, q_d, q_e)\)-CCA security for KEM having access to oracles ODECAP and OCHALL using an adversary \(A\) against \(G_0/G_1\) of the proof of Theorem 1.

| B([-\(\Gamma_{pak}\),OCHALL](\(pk_1,\ldots, pk_n\)) | Oracle CHALL(i \in [n], j \in [n], m, k, nonce, aad, info)
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01 for (i \in [n])</td>
<td>19 if (</td>
</tr>
<tr>
<td>02 for (j \in [n])</td>
<td>20 return (\bot)</td>
</tr>
<tr>
<td>03 (psk_{ij} \leftarrow \text{GenPSK})</td>
<td>21 ((c_1, K) \leftarrow \text{KEM Encaps}(pk_j))</td>
</tr>
<tr>
<td>04 (pk_{ij} \leftarrow psk_{ij})</td>
<td>22 if (j \notin \Gamma_{pak})</td>
</tr>
<tr>
<td>05 (E, \Gamma_{pak}, \Gamma_{pak} \leftarrow \emptyset)</td>
<td>23 (K \leftarrow \text{OCHALL}(j))</td>
</tr>
<tr>
<td>06 (b \leftarrow {0, 1})</td>
<td>24 ((k, nonce) \leftarrow \text{KS}(K, psk_{ij}, info))</td>
</tr>
<tr>
<td>07 (b' \leftarrow A_{Enc,Enc,Dec,Chall,RepPK,RepPSK}(pk_1,\ldots, pk_n))</td>
<td>25 (c_2 \leftarrow \text{AEAD Enc}(k, m, aad, nonce))</td>
</tr>
<tr>
<td>08 return ([b = b'])</td>
<td>26 (E \leftarrow E \cup {(pk_j, psk_{ij}, (c_1, c_2), aad, info)})</td>
</tr>
<tr>
<td>Oracle ENC</td>
<td>(c_1, K) \leftarrow \text{KEM Encaps}(pk_j)</td>
</tr>
</tbody>
</table>
| 09 \((k, nonce) \leftarrow \text{KS}(K, psk_{ij}, info)\) | Oracle REP PK(j \in [n], pk)
| 10 \(c_2 \leftarrow \text{AEAD Enc}(k, m, aad, nonce)\) | 28 \(pk_j \leftarrow pk\) |
| 11 return \((c_1, c_2)\) | 29 \(\Gamma_{pak} \leftarrow \Gamma_{pak} \cup \{j\}\) |
| Oracle DEC|(c_1, K) \leftarrow \text{OCHALL}(j, c_1) | Oracle REP PS K(i \in [n], j \in [n], m, nonce, info)
| 13 if \(j \in \Gamma_{pak} \lor (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in E\) | 30 \(psk_{ij} \leftarrow pk\) |
| 14 return \(\bot\) | 31 \(psk_{ij} \leftarrow pk\) |
| 15 \(K \leftarrow \text{ODecaps}(c_1)\) | 32 \(\Gamma_{pak} \leftarrow \Gamma_{pak} \cup \{(i, j), (j, i)\}\) |
| 16 \((k, nonce) \leftarrow \text{KS}(K, psk_{ij}, info)\) | |
Listing 23: Adversary $C_1$ against PRF security for $KS_1$ having access to oracle $Eval$ and using an adversary $A$ against $G_1/G_2$ of the proof of Theorem 1.

\[
\begin{align*}
C^{PSK} & \quad \text{Oracle $Enc(i \in [n], j \in [n], m, aad, info)$} \\
01 & \quad \text{for } i \in [n] \\
02 & \quad (sk_i, pk_i) \triangleq Gen \\
03 & \quad \text{for } j \in [n] \\
04 & \quad psk_{ij} \triangleq GenPSK \\
05 & \quad \text{Oracle $Encaps(pk_{ij})$} \\
06 & \quad C, E', \Gamma_p, i_pak \leftarrow 0 \\
07 & \quad i \leftarrow 0 \\
08 & \quad b \leftarrow \{0, 1\} \\
09 & \quad b' \leftarrow A^{Enc, Dec, Chall, RepPK, RepPSK}(pk_1, \ldots, pk_n) \\
10 & \quad \text{return } [b = b'] \\
\end{align*}
\]

in $Chall$. All together, we obtain

$$|Pr[G_2 \Rightarrow 1] - Pr[G_3 \Rightarrow 1]| \leq \frac{q_e^2 + q_c^2 + q_e q_c}{2^q}.$$

Game $G_4$. We modify $G_4$ by changing the output of $KS$ in every oracle in case of a non-replaced pre-shared key, i.e. $(i, j) \notin \Gamma_{\text{pask}}$, to uniformly random values (Line 19 for $Enc$, Line 38 for $Dec$, and Line 59 for $Chall$). The difference can be turned into a PRF adversary $C_2$ against $KS_2$, i.e. $KS$ keyed in the second component:

$$|Pr[G_3 \Rightarrow 1] - Pr[G_4 \Rightarrow 1]| \leq Adv_{c_2, KS_2}^{q_e + q_d + q_e q_d + q_e q_d}.$$  

Note that for the challenge oracle, the assumption $(i, j) \notin \Gamma_{\text{pask}}$ also implicitly holds. Since the change is only applied in the else case (Line 52) the receiver’s key was replaced, i.e. $j \in \Gamma_{\text{pask}}$. Additionally it is impossible that both $j \in \Gamma_{\text{pask}}$ and $(i, j) \in \Gamma_{\text{pask}}$ because in this case the challenge oracle would have returned $\bot$ due to the condition in Line 42. Thus, it holds $(i, j) \notin \Gamma_{\text{pask}}$ in the else case which means that $psk_{i,j}$ was not replaced and therefore uniformly sampled during the key generation. We construct adversary $C_2$ formally in Listing 24. For the pre-shared keys, the number of different keys is determined by the number of users. However, we need at most $q_e + q_d + q_e$ different PRF keys for the reduction and at most the same number of queries to the evaluation oracle.

**Reduction to Game $G_4$.** Game $G_4$ can be reduced to the CCA security of AEAD. We construct an adversary $D$ against CCA security of AEAD using adversary $A$ from $G_4$ as
Listing 24: Adversary $C_2$ against PRF security for $K_{S_2}$ having access to oracle $Eval$ and using an adversary $A$ against $G_3/G_4$ of the proof of Theorem 1.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:01</td>
<td>for $i \in [n]$</td>
</tr>
<tr>
<td>00:02</td>
<td>$(sk_i, pk_i) \leftarrow Gen$</td>
</tr>
<tr>
<td>00:03</td>
<td>for $j \in [l]$</td>
</tr>
<tr>
<td>00:04</td>
<td>$pk_{ij} \leftarrow GenPSK$</td>
</tr>
<tr>
<td>00:05</td>
<td>$pk_{ij} \leftarrow pk_{ij}$</td>
</tr>
<tr>
<td>00:06</td>
<td>$(E, E', \Gamma_{pk}, \Gamma_{sk}) \leftarrow \emptyset$</td>
</tr>
<tr>
<td>00:07</td>
<td>$l \leftarrow 0$</td>
</tr>
<tr>
<td>00:08</td>
<td>$b \leftarrow (0, 1)$</td>
</tr>
<tr>
<td>00:09</td>
<td>if $b \leftarrow A^{Enc, Dec, Chall, RepPSK, RepPSK}(pk_1, \ldots, pk_n)$</td>
</tr>
<tr>
<td>00:10</td>
<td>return $[b = b']$</td>
</tr>
</tbody>
</table>

Oracle $Enc(i \in [n], j \in [m], aad, info)$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02:01</td>
<td>$(c_1, K) \leftarrow KEM.Enc(pk_i)$</td>
</tr>
<tr>
<td>02:02</td>
<td>$(k, nonce) \leftarrow KSK(pk_{ij}, info)$</td>
</tr>
<tr>
<td>02:03</td>
<td>if $(i, j) \notin I_{pk}$</td>
</tr>
<tr>
<td>02:04</td>
<td>$(k', nonce', \ell') : (k', nonce', K, i, j, \ell, info) \in A_{Dec}$</td>
</tr>
<tr>
<td>02:05</td>
<td>$(k, nonce) \leftarrow (k', nonce')$</td>
</tr>
<tr>
<td>02:06</td>
<td>$Enc(\ell, K)[info] \leftarrow$ evaul query</td>
</tr>
<tr>
<td>02:07</td>
<td>$A_{Dec} \leftarrow A_{Dec} \cup {(k, nonce, K, i, j, \ell, info)}$</td>
</tr>
<tr>
<td>02:08</td>
<td>$c_1 \leftarrow A^{Enc}(k, m, aad, nonce)$</td>
</tr>
<tr>
<td>02:09</td>
<td>return $(c_1, c_2)$</td>
</tr>
</tbody>
</table>

Oracle $Dec(i \in [n], j \in [m], c_1, c_2, aad, info)$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04:01</td>
<td>if $(sk_i, sk_j) = \emptyset \cup {pk_{ij}, c_1, c_2, aad, info} \in E$</td>
</tr>
<tr>
<td>04:02</td>
<td>return $\bot$</td>
</tr>
<tr>
<td>04:03</td>
<td>$K \leftarrow KEM.DecEnc(sk_i, c_1)$</td>
</tr>
<tr>
<td>04:04</td>
<td>$(k, nonce) \leftarrow KS(K, pk_{ij}, info)$</td>
</tr>
<tr>
<td>04:05</td>
<td>if $\exists K' : (pk_{ij}, c_i, K') \in E'$</td>
</tr>
<tr>
<td>04:06</td>
<td>$(k, nonce) \leftarrow KS(K', pk_{ij}, info)$</td>
</tr>
<tr>
<td>04:07</td>
<td>if $\exists (k', nonce', K' \in [0, 1]^{ nonce})$</td>
</tr>
<tr>
<td>04:08</td>
<td>$(k, nonce) \leftarrow (k', nonce')$</td>
</tr>
<tr>
<td>04:09</td>
<td>$A_K \leftarrow A_K \cup {(k, nonce, K, pk_{ij}, info)}$</td>
</tr>
<tr>
<td>04:10</td>
<td>else if $(i, j) \notin I_{pk}$</td>
</tr>
<tr>
<td>04:11</td>
<td>$(k', nonce', K, i, j, \ell', info) \in A_{Dec}$</td>
</tr>
<tr>
<td>04:12</td>
<td>$(k, nonce) \leftarrow (k', nonce')$</td>
</tr>
<tr>
<td>04:13</td>
<td>$A_{Dec} \leftarrow A_{Dec} \cup {(k, nonce, K, i, j, \ell', info)}$</td>
</tr>
<tr>
<td>04:14</td>
<td>$c_2 \leftarrow AEAD.Enc(k, m, aad, nonce)$</td>
</tr>
<tr>
<td>04:15</td>
<td>$E \leftarrow E \cup {(pk_{ij}, pk_{ij}, c_1, c_2), aad, info}$</td>
</tr>
<tr>
<td>04:16</td>
<td>return $(c_1, c_2)$</td>
</tr>
</tbody>
</table>

Oracle $RepPK(j \in [n], pk)$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>06:01</td>
<td>$(sk_j, pk_j) := (\bot, pk)$</td>
</tr>
<tr>
<td>06:02</td>
<td>$I_{pk} \leftarrow I_{pk} \cup {(j, i)}$</td>
</tr>
</tbody>
</table>

Oracle $RepPSK(i \in [n], j \in [n], pk)$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>07:01</td>
<td>$pk_{ij} := pk$</td>
</tr>
<tr>
<td>07:02</td>
<td>if $j \notin I_{pk}$</td>
</tr>
<tr>
<td>07:03</td>
<td>$I_{pk} := I_{pk} \cup {(i, j)}$</td>
</tr>
</tbody>
</table>

The Pre-Shared Key Modes of HPKE 37

---

described in Listing 25. Adversary $D$ simulates the oracles using their own encryption oracle $Enc_{AED}$ and decryption oracle $Dec_{AED}$. The previous game modifications ensure that the keys and nonces for the AED encryptions are uniformly random and that there is no collision between oracle $Enc$ and $Chall$ which are both calling the encryption function. This matches the conditions of the CCA game for AED. The difficulty is that the simulator has to recognize if a query to the decryption oracle corresponds to the same key and nonce as for a previous call without knowing the actual key and nonce. This can be solved by keeping track of the queried parameter since $KS$ deterministically relates their inputs and outputs (set $E'$ and sets $A_{Enc}$ and $A_{Dec}$ from the previous games).
It remains to show that the simulation of the decryption oracle is sound. This is the case if the CCA decryption oracle is never queried on challenge parameters, i.e. on tuples \((\ell, c_2, aad)\) corresponding to a CCA encryption query (see Line 11 in the CCA experiment for AEAD). Assume that it is queried on such a tuple, i.e. \((\ell, c_2, aad)\) was part of a challenge query. However, since the inputs to KS must also be equal (condition in Line 11 and one-to-one relation from instance \(\ell\) to challenge query), values \(K, psk_{i,j}\), and info also match the challenge query. If \(c_1\) also equals the one from the challenge query, all relevant parameters are the same and the parameters can be found in set \(E\). However, in this case the decryption oracle aborts after the first check in Line 25, returns \(\bot\) and the CCA oracle is not queried. The remaining case is a query for a \(c_1\) which is not equal to the one stored in \(E\). This case can also not occur due to the condition in Line 25 which guarantees that the public key was not replaced \((sk_j \neq \bot)\) because otherwise the oracle returns \(\bot\) again. If the public key was not replaced, there exists an entry in set \(E'\) connecting \(K\) and \(c_1\) which means that we have the same \(c_1\) whenever we have the same \(K\) as for a previous challenge query. In total, the decryption oracle of the CCA game never returns \(\bot\) and the simulation is sound which completes the proof:

\[
\Pr[G_4 \Rightarrow 1] \leq \text{Adv}^{(qe+qd,qc)}_{D,\text{AEAD}}.\]

Putting everything together, we obtain the stated bound. \(\Box\)

### B.2 Proof of Theorem 2

**Proof.** We describe several games depicted in Listing 26.

**Game \(G_0\).** Let \(G_0\) be the \((n, q_e, q_d, q_c, r_{pk}, r_{sk}, r_{psk})\)-Insider-CCA game for the construction pskAPKE[AKEM, KS, AEAD], i.e.,

\[
\Pr[G_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, q_c, r_{pk}, r_{sk}, r_{psk})\text{-Insider-CCA}(A) \Rightarrow 1].
\]

**Game \(G_1\).** In this game, we make the following changes. If the challenge oracle is queried on a receiver index \(j\) which was not replaced, i.e. \(j \notin \Gamma_{pk}\), we replace the KEM key \(K\) by a uniformly random value from the key space \(K\). The result is then stored together with the receiver’s public key and the ciphertext in set \(E'\). We further change the decryption oracle by replacing the KEM key \(K\) by the one from set \(E'\) if there already exists such a key for the queried ciphertext \(c_1\) and the receiver’s public key \(pk_j\). Distinguishing these changes can be turned into an adversary \(B\) against \((n, q_d, q_{c1})\)-CCA security of KEM as depicted in Listing 27.

\[
|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \text{Adv}^{(n,q_q,q_d,q_{c1},r_{psk})}_{B,\text{AKEM}}.\]

Note that \(B\) can simulate all oracles by their own or with the oracles provided by their own experiment. For RepPK and RepSK, the simulator can store the values and answer other oracle queries accordingly. Note that we introduce another set \(R\) to store the indices for which the public key was replaced to differentiate public key replacements from secret key replacements since we handle them differently. \(^6\) The encryption and decryption oracle can

\(^6\) This is implicitly done in the security notion, e.g. decryption queries are still possible for replaced receiver’s secret keys but not for replaced receiver’s public keys since the experiment is not able to decrypt anymore.
be simulated by the encapsulation and decapsulation oracle provided by the AKEM Insider-CCA game. If the challenge oracle is queried on non-replaced receiver keys, i.e. \( j \notin I_{pk} \) (Line 24), \( B \) can use their own challenge. If \( B \) is in the real case of their \((n,q_d,q_c)\)-CCA game, they perfectly simulate \( G_0 \). If they are in the random case, they perfectly simulate \( G_1 \).

**Game \( G_2 \)**. Game \( G_1 \) is exactly Game \( G_1 \) in the proof of Theorem 1. Hence, we can apply the same game modifications as in the previous proof to obtain the same final game, called
Listing 26: Games $G_0 - G_1$ for the proof of Theorem 2.

<table>
<thead>
<tr>
<th>$G_0 - G_1$</th>
<th>Oracle CHALL($i \in [n], j \in [n], m_0, m_1, aad, info$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 for $i \in [n]$</td>
<td>24 if $</td>
</tr>
<tr>
<td>02 $(sk_i, pk_i) \xleftarrow{} \text{Gen}$</td>
<td>25 return $\bot$</td>
</tr>
<tr>
<td>03 for $j \in [n]$</td>
<td>26 $(c_1, K) \xleftarrow{} \text{AuthEncap}(sk_i, pk_i)$  // $G_1$</td>
</tr>
<tr>
<td>04 $psk_{ij} \xleftarrow{} \text{GenPSK}$</td>
<td>27 if $j \notin I_{pk}$</td>
</tr>
<tr>
<td>05 $psk_{ij} \leftarrow psk_{ij}$</td>
<td>28 $K \xleftarrow{} \bot$  // $G_1$</td>
</tr>
<tr>
<td>06 $E, E', I_{pk}, I_{pak} \leftarrow \emptyset$</td>
<td>29 $E' \leftarrow E' \cup {(pk_i, pk_j, c, K)}$  // $G_1$</td>
</tr>
<tr>
<td>07 $b' \xleftarrow{\text{Enc}, \text{Dec}, \text{Chall}, \text{RepPK}, \text{RepPSK}}$</td>
<td>30 $(k, nonce) \leftarrow \text{KS}(K, psk_{ij}, info)$</td>
</tr>
<tr>
<td>08 return $[b = b']$</td>
<td>31 $c_2 \leftarrow \text{AEAD.Enc}(k, m_0, aad, nonce)$</td>
</tr>
<tr>
<td>Oracle $\text{Enc}(i \in [n], j \in [n], m, aad, info)$</td>
<td>32 $E \leftarrow E \cup {(pk_i, pk_j, psk_{ij}, (c_1, c_2), aad, info)}$</td>
</tr>
<tr>
<td>10 if $sk_i = \bot$</td>
<td>33 return $(c_1, c_2)$</td>
</tr>
<tr>
<td>11 return $\bot$</td>
<td>Oracle $\text{RepPK}(j \in [n], pk)$</td>
</tr>
<tr>
<td>12 $(c_1, K) \xleftarrow{} \text{AuthEncap}(sk_i, pk_i)$</td>
<td>34 $(sk_i, pk_i) \leftarrow (pk_i, \bot)$</td>
</tr>
<tr>
<td>13 $(k, nonce) \leftarrow \text{KS}(K, psk_{ij}, info)$</td>
<td>35 $I_{pk} \leftarrow I_{pk} \cup {}$</td>
</tr>
<tr>
<td>14 $c_2 \leftarrow \text{AEAD.Enc}(k, m, aad, nonce)$</td>
<td>Oracle $\text{RepSK}(j \in [n], sk)$</td>
</tr>
<tr>
<td>15 return $(c_1, c_2)$</td>
<td>36 $(sk_i, pk_i) \leftarrow (sk_i, \mu(sk))$</td>
</tr>
<tr>
<td>Oracle $\text{Dec}(i \in [n], j \in [n], (c_1, c_2), aad, info)$</td>
<td>37 $I_{pk} \leftarrow I_{pk} \cup {}$</td>
</tr>
<tr>
<td>16 if $sk_i = \bot \lor (pk_j, psk_{ij}, (c_1, c_2), aad, info) \in E$</td>
<td>Oracle $\text{RepPSK}(i \in [n], j \in [n], psk)$</td>
</tr>
<tr>
<td>17 return $\bot$</td>
<td>38 $psk_{ij} \leftarrow psk$</td>
</tr>
<tr>
<td>18 if $\exists K : (pk_i, pk_j, c, K) \in E'$  // $G_1$</td>
<td>39 $psk_{ij} \leftarrow psk$</td>
</tr>
<tr>
<td>19 return $K$  // $G_1$</td>
<td>40 $I_{pak} \leftarrow I_{pak} \cup {(i,j),(j,i)}$</td>
</tr>
<tr>
<td>20 $K \leftarrow \text{AuthDecap}(pk_i, sk_j, c_1)$</td>
<td></td>
</tr>
</tbody>
</table>
Listing 27: Adversary $B$ against $(n, q_e, q_d, q_c, r_{pk}, r_{sk})$-Insider-CCA security for AKEM with access to oracles OAEncap, OADecap, OChall, and ORepSK. Adversary $A$ against $G_0/G_1$ makes at most $q_e$ queries to Enc, at most $q_d$ queries to Dec, at most $q_c$ queries to Chall, at most $r_{pk}$ queries to RepPK, at most $r_{sk}$ queries to RepSK, and at most $r_{psk}$ queries to RepPSK.

\[
\text{Oracle Chall}(i \in [n], j \in [n], m_0, m_1, \text{aad, info})
\]

\[
\text{Oracle RepPK}(j \in [n], pk)
\]

\[
\text{Oracle RepPSK}(j \in [n], psk)
\]
G_2 This results in the same bound

\[ \Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1] \leq \text{Adv}_{C_1,K}^{(q_{e}, q_{d} + q_{e})} + \text{Adv}_{C_2,K}^{(q_{e} + q_{d} + q_{e} + q_{d})} \]

with a final winning probability of

\[ \Pr[G_2 \Rightarrow 1] = \text{Adv}_{EAD}^{(q_{e} + q_{d})}. \]

Putting everything together results in the stated bound.

**B.3 Proof of Theorem 3**

**Proof.** We describe a series of games in Listing 28.

**Listing 28:** Games G_0 – G_7 for the proof of Theorem 3.

**Game G_0.** This is the (n, q_e, q_d, r_{psk}, r_{psk})-Auth game for pskPKE|KEM, KS, AEAD], thus we have

\[ \Pr[G_0 \Rightarrow 1] = \Pr[(n, q_e, q_d, r_{psk}, r_{psk})-\text{Auth}(\Lambda) \Rightarrow 1]. \]

**Game G_1.** We introduce a set A to log the outputs of KS produced in Enc and Dec. More specifically, A contains elements k, nonce, i, j, K, psk, info of the form that k, nonce is the output of a KS call, K, psk, info, the input, and i, j sender and receiver index for
that query. If $KS$ was already called with the same parameters, we can take the stored value. For the encryption oracle, this is only done for queries for which the corresponding receiver has not been corrupted, i.e. for queries with $sk_j \neq \bot$ (Line 11). For the decryption oracle, we use the stored value only if the corresponding $psk$ has not been replaced, i.e. for queries on indices $(i, j) \notin \Gamma_{psk}$ (Line 29). Since the change is only of formal nature, we have

$$\Pr[G_0 \Rightarrow 1] = \Pr[G_1 \Rightarrow 1].$$

**Game $G_2$.** If the receiver’s key was not replaced, we replace the KEM secret $K$ by a uniformly random value in the encryption oracle $Enc$ The result is stored to answer the decryption oracle correctly. The change corresponds to an $(n, q_d, q_e)$-CCA adversary against KEM. For simulating the $RepPK$ oracle, we refer to the proof of Theorem 1 which presents a solution for the same case. We obtain

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \text{Adv}^{(n, q_d, q_e)}_{\text{KEM}}.$$  

**Game $G_3$.** We abort in the encryption oracle if the corresponding receiver’s key was not replaced and there already exists an entry in $\Lambda$ with the queried parameters. Since $K$ was randomly chosen in the previous game, the probability of having such an entry is at most $|\Lambda|/|K|$. Note that $\Lambda$ is filled with an additional element at most once per $Enc/Dec$ query. This yields the following advantage:

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \frac{q_e(q_e + q_d - 1)}{|\Lambda|}.$$  

**Game $G_4$.** If the receiver’s key was not replaced and we do not abort, we replace the output of $KS$ in the encryption oracle with uniformly random values of the respective domain. The changes in Game $G_1$ ensure consistent outputs of $KS$, i.e. queries on $Enc$ or $Dec$ with the same parameters lead to the same output of $KS$. Hence, games $G_3$ and $G_4$ can only be distinguished by distinguishing the real output of $KS$ from a uniformly random one. This can be turned into an adversary against PRF security of $KS_1$, i.e. keyed on the first input. Note that $K$ is chosen uniformly at random due to the changes in Game $G_2$. There are at most $q_e$ different instances for the PRF and at most the same number of queries resulting in

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \text{Adv}^{(q_e)}_{\text{PRF}_{c_1}, KS_1}.$$  

**Game $G_5$.** Now we apply changes to the decryption oracle analogously to the changes from the previous game. If the corresponding $psk$ has not been revealed, we replace the output of $KS$ in the decryption oracle with uniformly random values of the respective domain. Again, we have consistent outputs of $KS$, i.e. queries on $Enc$ and $Dec$ with the same parameters lead to the same output of $KS$. Games $G_4$ and $G_5$ can only be distinguished by distinguishing the real output of $KS$ from a uniformly random one. This can be turned into an adversary against PRF security of $KS_2$, i.e. keyed on the second input. Note that $psk$ was not replaced and is therefore uniformly random which matches the PRF game. There are at most $q_d$ different instances for the PRF and at most the same number of queries resulting in

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq \text{Adv}^{(q_d)}_{\text{PRF}_{c_2}, KS_2}.$$
Game $G_6$. We now store the results of encryption queries to replace the actual decryption in the case of an honest oracle query, i.e. $(i, j) \not\in \Gamma_{\text{psk}}$, by the stored results. This is only a formal change since the correctness of the AEAD implies same win probabilities for this game change:

$$\Pr[G_5 \Rightarrow 1] = \Pr[G_6 \Rightarrow 1].$$

Game $G_7$. In this game, we replace the actual decryption in an honest decryption oracle query with $\bot$. Distinguishing the game difference can be turned into an adversary against INT-CTXT security of the underlying AEAD. We depict the adversary in Listing 29. There are $q_e + q_d$ instances and adversary $D_1$ makes at most $q_d$ queries to their decryption oracle $\text{Dec}_{\text{AEAD}}$. Thus, we obtain

$$|\Pr[G_6 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq \text{Adv}_{D_1,\text{AEAD}}^{(q_e + q_d, q_d)\text{-INT-CTXT}}.$$

Reduction to Game $G_7$. We can reduce INT-CTXT security to Game $G_7$. An adversary against INT-CTXT can be used to simulate $G_7$. The adversary can simulate the decryption oracle since in cases $(i, j) \not\in \Gamma_{\text{psk}}$, they can output $\bot$ (or the original encryption if it was produced during the experiment). In cases $(i, j) \in \Gamma_{\text{psk}}$, they can compute the output by their own. For the encryption oracle, the adversary can compute the output on their own in cases $sk_j = \bot$ and use their own encryption oracle in cases $sk_j \neq \bot$ since key $k$ and nonce are uniformly random similarly to the adversary in Listing 29.

The output of the adversary against game $G_7$ can then be used to issue a decryption query in the INT-CTXT experiment on either a new instance or a previous instance if the output parameters $i^*, j^*, c_1^*, \text{info}^*$ match with that instance. Matching parameters can be identified by computing $K^* \leftarrow \text{Decaps}(sk_{j^*}, c_1^*)$ and comparing $(i^*, j^*, K^*, psk_{j^*}, \text{info}^*)$ to set $\Lambda$ similar to Line 14 or Line 31 in Listing 29. If the adversary against $G_7$ wins, the adversary against the INT-CTXT experiment has a valid ciphertext which does not decrypt to $\bot$ due to the winning condition of $G_7$. Further, since the ciphertext must not decrypt to $\bot$, it also holds $sk_{j^*} \neq \bot$ which means that the parameters correspond to an actual instance of the simulator. That means the adversary can distinguish between the real or random case since the result of the decryption query must be unequal to $\bot$ in the real case. This results in

$$\Pr[G_7 \Rightarrow 1] \leq \text{Adv}_{D_2,\text{AEAD}}^{(q_e + 1, 1)\text{-INT-CTXT}}.$$

Putting everything together we obtain the stated bound.

\[\square\]

C Proofs for the Hybrid Post-Quantum APKE

C.1 Proof of Theorem 5

Proof. In order to prove Theorem 5, we will prove the following two inequalities.

\[
\text{Adv}_{A,\text{APKE}[\text{AEKEM}_1, \text{AEKEM}_2, \text{AEAD}]_{\Lambda}}^{(n, q_e, q_d, q_c)}\text{-Insider-CCA} \leq \text{Adv}_{B,\text{AEKEM}_1}^{(n+1, q_e, q_d, q_c)\text{-Insider-CCA}} + \text{Adv}_{C,\text{KS}_1}^{(q_e, q_d, q_c)\text{-PRF}} + \text{Adv}_{D,\text{AEAD}}^{(q_e, q_d)\text{-CCA}}
\]  

(1)
Then, we can apply a similar proof as for Theorem 2 with the following change. Due to
the collisions as a preparation for the security loss from querying the special structure of adversary
and

\[
\text{RepPK}(10^9 \ell \rightarrow 0^9 (12^3 K \leftarrow 0^3 (17^6 \ell \leftarrow 0^6 (22^0 \text{Oracle} \leftarrow 24^0 \text{RepPK})))}
\]

Oracle Dec(i ∈ [n], j ∈ [n], (c1, c2), aad, info)

\[
\text{Enc}(c(i) \leftarrow \text{Encaps}(pk_i)) \\
\text{if sk_j ≠ ⊥} \\
K \leftarrow \hat{K} \\
\hat{E} \leftarrow \hat{E} \cup \{pk_j, c1, K\} \\
\text{if } \exists \ell : ((i, j, K, pk_{ij}, info) \in A) \\
\text{abort} \\
\ell \leftarrow \ell + 1 \\
c2 \leftarrow \text{Enc}(c(i) \leftarrow \text{Encaps}(pk_i)) \\
\text{else} \\
(k, nonce) \leftarrow \text{KS}(K, pk_{ij}, info) \\
\ell \leftarrow \ell + 1 \\
c2 \leftarrow \text{Enc}(c(i) \leftarrow \text{Encaps}(pk_i, (c1, c2), aad, info)) \\
\text{return } (c1, c2)
\]

Oracle \(\text{Dec}(i \in [n], j \in [n], (c1, c2), aad, info)\)

25 if \(\text{sk}_j \leftarrow \bot\) 
26 return \(⊥\) 
27 \(K \leftarrow \text{Decaps}(sk_j, c1)\) 
28 if \(\exists K' : (pk_j, c1, K') \in \hat{E}\) 
29 \(K \leftarrow K'\) 
30 if \((i, j) \notin I_{psk}\) 
31 if \(\exists \ell : ((i, j, K, pk_{ij}, info) \in A)\) 
32 \(m \leftarrow \text{Dec}(c(i) \leftarrow \text{Encaps}(pk_j, c2, aad))\) 
33 \text{else} 
34 \(\ell \leftarrow \ell + 1\) 
35 \(m \leftarrow \text{Dec}(c(i) \leftarrow \text{Encaps}(pk_j, c2, aad))\) 
36 \text{else} 
37 \((k, nonce) \leftarrow \text{KS}(K, pk_{ij}, info)\) 
38 \(m \leftarrow \text{Dec}(c(i) \leftarrow \text{Encaps}(pk_j, c2, aad, nonce))\) 
39 \(A \leftarrow A \cup \{(i, j, K, pk_{ij}, info)\}\) 
40 return \(m\)

Oracle \(\text{RepPK}(j \in [n], pk)\)

41 \((sk_j, pk_j) \leftarrow ((\bot, pk))\)

Oracle \(\text{RepPSK}(i \in [n], j \in [n], psk)\)

42 \(pk_{ij} \leftarrow \text{psk}\) 
43 \(pk_{ij} \leftarrow \text{psk}\) 
44 \(I_{psk} \leftarrow I_{psk} \cup \{(i, j, (i, j))\}\)

and

\[
\text{Adv}^{(n, q_e + q_d, q_e)}_{\mathcal{A}, \text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{KS}, \text{AEAD}]} \leq \text{Adv}^{(n, q_e + q_d, q_e)}_{\mathcal{A}, \text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{KS}, \text{AEAD}]} + \text{Adv}^{q_e + q_d + q_e + q_{psk}}_{\mathcal{A}, \text{PSKDec}[\text{KS}_2, \text{AEAD}]} + \text{Adv}^{q_e + q_d + q_e + q_{psk}}_{\mathcal{A}, \text{PSKDec}[\text{KS}_2, \text{AEAD}]}.
\]

Inequality (1): First, we show

\[
\text{Adv}^{(n, q_e + q_d, q_e)}_{\mathcal{A}, \text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{KS}, \text{AEAD}]} \leq \text{Adv}^{(n + 1, q_e + q_d, q_e + q_d, q_e + q_{psk})}_{\mathcal{A}, \text{pskAPKE}[\text{AKEM}, \text{KS}, \text{AEAD}]}.
\]

Then, we can apply a similar proof as for Theorem 2 with the following change. Due to
the special structure of adversary \(\mathcal{A}\), we can skip the game which introduces the PRF security loss from querying \(\text{KS}\) in the second input (the \(\text{psk}\) part). This is because none of the conditions which are used for the PRF advantage are met. In detail, for the challenge oracle, it always holds \(j \notin I_{psk}\) since the challenge oracle may only be queried for an honest receiver. The same holds for the loss based on the entropy of the KEM key which eliminates collisions as a preparation for the \(\text{psk}\) part of the PRF and the number of different keys in the CCA experiment for \(\text{AEAD}\). Due to the structure of the adversary in Listing 30 it
To prove (3), we use an $\text{APKE}$ but only if there is a corresponding element in uniformly random values. Additionally, the decryption oracle is modified in the same way.

This can be directly turned into an adversary against $\text{CCA}$.

To prove (3), we use an $(n, q, q_d, q_e, q_c)$-$\text{APKE}$ adversary against $\text{APKE}$ to construct an adversary $\mathcal{A}$ against $(n + 1, q, q_d, q_e, q_c, q_d + q_e)$-$\text{APKE}$ as shown in Listing 30. One can see that $\mathcal{A}$ uses one query to $\text{RepPK}$ for each query to $\text{Enc}$ and $\text{pskDec}$, i.e. $r_{pk} = q_e + q_d$, and one query to $\text{RepPSK}$ for each $\text{Chall}$ query, i.e. $r_{sk} = q_c$. Additionally, we need to replace the “$\text{psk}$” for each query resulting in $r_{psk} = q_e + q_d + q_c$. This fulfills the adversarial query bounds from inequality (3).

Listing 30: $(n + 1, q, q_d, q_e, q_c, q_d + q_e)$-$\text{APKE}$ adversary $\mathcal{A}$ against $\text{pskAPKE}[\text{AKEM}_1, \text{KS}, \text{AEAD}]$ using an $(n, q, q_d, q_c)$-$\text{APKE}$ adversary $\mathcal{A}$ against $\text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{KS}, \text{AEAD}]$ for Inequality (3) of the proof of Theorem 5.

| 01 for $i \in [n]$ |
| 02 $(sk_1^i, pk_1^i) \leftarrow \text{Gen}$ |
| 03 $pk_i \leftarrow (pk_1^i, pk_2^i)$ |
| 04 $b' \leftarrow \mathcal{A}^{\text{Enc}, \text{Dec}, \text{Chall}}((pk_1, \ldots, pk_n)$ |
| 05 return $b$ |

Oracle $\text{AEnc}(i \in [n], (pk_1^i, pk_2^i), m, \text{aad, info})$:

06 if $\exists j' \in [n] : pk_2^i = pk_1^{j'}$
07 $j \leftarrow j'$
08 else
09 $\text{ORepPK}(n + 1, pk_1^{j + 1})$
10 $j \leftarrow n + 1$
11 $(c', K') \leftarrow \text{AuthEncap}(sk_1^{j'}, pk_1^{j'})$
12 $\text{ORepPSK}(i, j, K')$
13 $c \leftarrow \text{Enc}(i, j, m, \text{aad, info})$
14 return $(c, c')$

Oracle $\text{AdDec}((pk_3^j, pk_2^j), j \in [n], (c', c''), \text{aad, info})$

15 if $\exists i' \in [n] : pk_1^{j} = pk_1^{i'}$
16 $i \leftarrow i'$
17 else
18 $\text{ORepPK}(n + 1, pk_1^{i + 1})$
19 $i \leftarrow n + 1$
20 $K' \leftarrow \text{AuthDecap}_2(pk_2^{i}, sk_1^{i'}, c'')$
21 $\text{ORepPSK}(i, j, K')$
22 $m \leftarrow \text{Dec}(i, j, c', \text{aad, info})$
23 return $m$

Oracle $\text{Chall}((sk_1^i, sk_2^j), j \in [n], m_0, m_1, \text{aad, info})$

24 $\text{ORepSK}(n + 1, sk_1^i)$
25 $(c', K')\text{AuthEncap}_2(sk_2^j, pk_2^j)$
26 $\text{ORepPSK}(n + 1, j, K')$
27 $c \leftarrow \text{Chall}(n + 1, j, m_0, m_1, \text{aad, info})$
28 return $(c, c')$

Inequality (2): Proving the second inequality (2), we start with the game $(n, q, q_d, q_e)$-$\text{CCA}$ for $\text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{KS}, \text{AEAD}]$ denoted by $G_0$ in Listing 31.

Game $G_1$. In Game $G_1$, we replace the $\text{AKEM}$ secret of $\text{AKEM}_2$ in the challenge oracle by a uniformly random value. This can be directly turned into an adversary against $\text{AKEM}_2$’s $\text{Insider-CCA}$ security:

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \text{Adv}_{\mathcal{A}, \text{AKEM}_2}^{(n, q, q_d, q_e)}.$$  

Game $G_2$. In $G_2$, the challenge oracle is changed such that the output of $\text{KS}$ is replaced by uniformly random values. Additionally, the decryption oracle is modified in the same way but only if there is a corresponding element in $\mathcal{E}'$. Since $K'$, which is the second key to $\text{KS}$,
which were sampled before. There are at most constructs an adversary

Reduction to Game

queries resulting in adversary against this can be turned into an PRF

is uniformly random due to the changes in the last game, this can be turned into an PRF adversary against $KS_2$, i.e. $KS$ keyed on the second input. Due to changes in the previous game, we keep consistent outputs with the decryption oracle since we reuse output values which were sampled before. There are at most $q_c$ different keys as well as $q_e$ evaluation queries resulting in

$$\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1] \leq \text{Adv}_{C,KS_2}^{(q_c,q_e) - \text{PRF}}.$$ 

Reduction to Game $G_2$. Game $G_2$ can be reduced to the CCA security of AEAD. We construct an adversary $D$ against CCA security of AEAD using adversary $A$ from $G_2$. This similar to the the reduction to Game $G_4$ in the proof of Theorem 1 and the adversary constructed in Listing 25. Note that there is a difference in the number of AEAD keys since we do not have a static $psk$ and adversary $D$ can simulate encryption queries without any query to their own oracles. This results in

$$\Pr[G_2 \Rightarrow 1] \leq \text{Adv}_{D,AEAD}^{(q_c,q_e) - \text{CCA}}.$$ 

Putting everything together, we obtain (2) which concludes the proof. □
C.2 Proof of Theorem 6

Proof. In order to prove Theorem 6, we prove the following two equations.

\[
\text{Adv}^{(n, q_e, q_d)}_{\mathcal{A}, \text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{AEAD}]} - \text{Adv}^{(n+1, q_e, q_d, q_e)}_{\mathcal{B}_1, \text{AKEM}_1} \\
+ \text{Adv}^{(n+1, q_e, q_d)}_{\mathcal{B}_2, \text{AKEM}_1} \\
+ \text{Adv}^{(g_q, q_i, q_e + q_d)}_{\mathcal{C}, \text{KS}_2} \\
+ \text{Adv}^{(2q_i + q_d + 1, q_e + 1)}_{\mathcal{D}, \text{AEAD}} \\
+ \frac{q_e(q_e + q_d - 1)}{|K|} \leq \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{A}, \ln(\epsilon/\tau)} - \text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{A}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{B}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{H}, \text{AEAD}}}{|\mathcal{C}|}
\]

and

\[
\text{Adv}^{(n, q_e, q_d)}_{\mathcal{A}, \text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{AEAD}]} - \text{Adv}^{(n+1, q_e, q_d, q_e)}_{\mathcal{B}_1, \text{AKEM}_2} \\
+ \text{Adv}^{(n+1, q_e, q_d)}_{\mathcal{B}_2, \text{AKEM}_2} \\
+ \text{Adv}^{(q_e, q_d)}_{\mathcal{C}, \text{KS}_2} \\
+ \text{Adv}^{(2q_i + q_d + 1, q_e + 1)}_{\mathcal{D}, \text{AEAD}} \\
+ \frac{q_e(q_e + q_d - 1)}{|K|} \leq \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{A}, \ln(\epsilon/\tau)} - \text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{A}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{B}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{H}, \text{AEAD}}}{|\mathcal{C}|}
\]

Inequality (4): We use a similar technique as in the proof of Theorem 5. First, we show

\[
\text{Adv}^{(n, q_e, q_d)}_{\mathcal{A}, \text{APKE}[\text{AKEM}_1, \text{AKEM}_2, \text{AEAD}]} - \text{Adv}^{(n+1, q_e, q_d, q_e, q_d)}_{\mathcal{A}, \text{pskAPKE}[\text{AKEM}_1, \text{KS}_2, \text{AEAD}]} \leq \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{A}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{B}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{H}, \text{AEAD}}}{|\mathcal{C}|}
\]

Then, we can apply a similar proof as for Theorem 4 with the following change. Due to the special structure of adversary \( \mathcal{A} \), we can skip the changes in Game G6 which introduces the PRF security loss from querying KS in the second input (the psk part). This is because none of the conditions which are used for the PRF advantage are met. In detail, for both the encryption and decryption oracle the conditions are \( i \notin \Gamma_{\text{psk}} \lor (i, j) \notin \Gamma_{\text{pk}} \) and \( i \in \Gamma_{\text{pk}} \) which leads to the condition \( (i, j) \notin \Gamma_{\text{psk}} \). However, \( (i, j) \notin \Gamma_{\text{psk}} \) never holds since \( \mathcal{A} \) queries the RePSK oracle before querying the encryption or decryption oracle.

This results in

\[
\text{Adv}^{(n+1, q_e, q_d, q_e, q_d)}_{\mathcal{A}_{\text{pskAPKE}[\text{AKEM}_1, \text{KS}_2, \text{AEAD}]} - \text{Adv}^{(n+1, q_e, q_d, q_e)}_{\mathcal{B}_1, \text{AKEM}_1} \\
+ \text{Adv}^{(n+1, q_e, q_d)}_{\mathcal{B}_2, \text{AKEM}_1} \\
+ \text{Adv}^{(q_e, q_d)}_{\mathcal{C}, \text{KS}_2} \\
+ \text{Adv}^{(2q_i + q_d + 1, q_e + 1)}_{\mathcal{D}, \text{AEAD}} \\
+ \frac{q_e(q_e + q_d - 1)}{|K|} \leq \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{A}, \ln(\epsilon/\tau)} - \text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{A}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{B}, \text{AEAD}}}{|\mathcal{C}|} + \frac{\text{Adv}^{(2, q_e + q_d + 1)}_{\mathcal{H}, \text{AEAD}}}{|\mathcal{C}|}
\]

To prove (6), we use an \( (n, q_e, q_d) - \text{Outider-Auth} \) adversary against APKE to construct an adversary \( \mathcal{A} \) against \( (n+1, q_e, q_d, q_e + q_d, q_e + q_d) - \text{Outider-Auth} \) security of pskAPKE as
Listing 32: $(n + 1, q_e, q_d, q_e + q_d, 0, q_e + q_d)$-Outsider-Auth adversary $\mathcal{A}$ against pkAKEM$_1$, KS, AEAD using an $(n, q_e, q_d)$-Outsider-Auth adversary $\mathcal{A}$ against APKE for Inequality (6) of Theorem 6.

![Listing 32](image)

shown in Listing 32. One can see that $\mathcal{A}$ uses one query to RepPK for each query to Enc and Dec, i.e. $r_{pk} = q_e + q_d$. Additionally, we need to replace the "psk" for each query resulting in $r_{psk} = q_e + q_d$. This fulfills the adversarial query bounds from inequality (6).

Inequality (5): Proving the second inequality (5), we start with the $(n, q_e, q_d)$-Outsider-Auth game for APKE using Games $G_0$ in Listing 33.

Listing 33: Games $G_0$ for the proof of Inequality (5) of Theorem 6.

![Listing 33](image)

Game $G_1$. We replace the AKEM secret of AKEM$_2$ by a uniformly random value in the encryption oracle and store the result together with the corresponding parameters in set
$E'$ so that the decapsulation gives consistent results (Line 25). The advantage can be turned into an adversary against the Outsider-CCA security of $\text{AKEM}_2$ issuing a challenge query for each query to $\text{ENC}$ and a decapsulation query for each query to $\text{pskDec}$. Note that there are no encapsulation queries necessary. This results in

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \text{Adv}^{(n, q_d, q_r)}_{\text{AKEM}_2} - \text{Outsider-CCA}.$$

**Game $G_2$.** Now we replace the AKEM secret of $\text{AKEM}_2$ by a uniformly random value in the decryption oracle if the parameter set was not queried before, i.e. the parameters do not occur in $E'$ which stores results from previous queries to $\text{AEnc}$ and $\text{ADec}$ (Line 24), the receiver is honest, i.e. $pk^2 \in \{pk_1^2, \ldots, pk_l^2\}$, and the shared KEM secret is not $\perp$. We additionally add a bit $b$ (Line 29) which indicates if the before mentioned case occurs which will be used in further game modifications. The setup now matches with oracles of an Outsider-Auth adversary against $\text{AKEM}_2$ and such an adversary can simulate the games. This results in the following advantage:

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \text{Adv}^{(n, q_d, q_r)}_{\text{Outsider-Auth}}.$$

**Game $G_3$.** We insert a set $\Lambda$ to log the outputs of $\text{KS}$. If $\text{KS}$ was already called with the same parameters, we take the stored value. Since the change is only conceptual due to a deterministic $\text{KS}$, we have

$$\Pr[G_2 \Rightarrow 1] = \Pr[G_3 \Rightarrow 1].$$

**Game $G_4$.** In case $b = 1$ (the case we replaced the KEM secret in Game $G_2$), we replace the output of $\text{KS}$ with uniformly random values of the respective domain. The changes in the last game ensure consistent outputs of $\text{KS}$, i.e. queries on $\text{AEnc}$ and $\text{ADec}$ with the same parameters lead to the same output of $\text{KS}$. Hence, game $G_3$ and $G_4$ can only be distinguished by distinguishing the real output of $\text{KS}$ from a uniformly random one. This can be turned into an adversary against PRF security of $\text{KS}_2$, i.e. keyed in the second input, since key $K'$ is uniformly chosen in case $b = 1$ due to the changes in Game $G_2$. There are at most $q_d$ different keys and at most $q_d$ different queries resulting in

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \text{Adv}^{(q_d, q_d)}_{\text{KS}_2} \text{PRF}.$$ 

**Game $G_5$.** We abort in the encryption oracle if there already exists an entry in $\Lambda$ with the queried parameters. Since $K'$ was randomly chosen in $G_1$, the probability of having such an entry is at most $|\Lambda|/|\mathcal{K}|$. Note that $\Lambda$ is filled with another element at most once per $\text{AEnc}/\text{ADec}$ query. This yields the following advantage:

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq \frac{q_e(q_e + q_d - 1)}{|\mathcal{K}|}.$$ 

**Game $G_6$.** In this game, we replace the actual decryption in an honest decryption oracle query, i.e. case $b = 1$, with $\perp$. The same holds if there is already an element in $\Lambda$ which has the same parameters. Distinguishing the game difference can be turned into an adversary against INT-CTX security of the underlying AEAD, the construction can be found in Listing 34. There are $q_e + q_d$ different keys and adversary $D_1$ makes at most $q_d$ queries to their decryption oracle $\text{Dec}_{\text{AEAD}}$. Thus, we obtain

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq \text{Adv}^{(q_e, q_d)}_{\text{Dec}_{\text{AEAD}}} - \text{INT-CTX}. $$
Reduction to Game $G_6$. We can reduce INT-CTX security to Game $G_6$ by constructing an adversary simulating $G_6$ very similar to the one in Listing 34. The adversary can simulate the decryption oracle since in case $b=1$ (Line 31 in Listing 33) and when there is a corresponding element in $A$ (Line 40 in Listing 33), they can output $\bot$ (or the original encryption if it was produced during the experiment). Otherwise, they can compute the output by their own. For the encryption oracle, the adversary can use their own encryption oracle similar to the adversary from the last game hop. The output of the adversary against game $G_6$ can then be used to issue a decryption query in the INT-CTX experiment on either a new instance or a previous instance if parameters $(pk^1_1, pk^2_1, (pk^1_2, pk^2_2), K^*, K'^*, c^*||info^*)$ match with that instance. Matching parameters can be identified by computing $K^* \leftarrow \text{AuthDecap}_1(sk_{j^*}, c_1)$, $K'^* \leftarrow \text{AuthDecap}_2(sk_{j^*}, c'^*)$ and comparing $((pk^1_1, pk^2_1), (pk^1_2, pk^2_2), K^*, K'^*, c^*||info^*)$ to set $A$ similar to Line 12 in Listing 34. If the adversary against $G_6$ wins, the adversary against the INT-CTX experiment has a valid ciphertext which does not decrypt to $\bot$ due to the winning condition of $G_6$. That means they can distinguish between the real or random case since the result of the decryption query must be unequal to $\bot$ in the real case. This results in

$$\Pr[G_6 \Rightarrow 1] \leq \text{Adv}^{(q_e+1,1)}_{D_2,\text{AEAD}}.$$ 

Putting everything together, we obtain the stated bound.

Listing 34: $(q_e + q_{ad}, q_d)$-INT-CTX adversary $D_1$ against AEAD using an adversary $A$ against Game $G_6$ from the proof of Inequality (5) of Theorem 6.

```
01 \textbf{D}_1^{\text{AEAD},\text{AEAD}}:
02 \textbf{for } i \in \{n\}
03 \quad (sk_i^1, pk_i^1) \leftarrow \text{AKEMI}().\text{Gen}
04 \quad (sk_i^2, pk_i^2) \leftarrow \text{AKEMI}.\text{Gen}
05 \quad pk \leftarrow (pk_i^1, pk_i^2)
06 \quad E, E', A \leftarrow \emptyset
07 \quad (*, *, (c', c''), \text{AAD}, \text{INFO}) \leftarrow A^{\text{AEAD},\text{AEAD}}((pk_1, \ldots, pk_n))
08 \quad \textbf{return } (E', A)

\textbf{Oracle $\text{AEAD}(c \in \{n\}, pk^i, pk^j)$, $m, \text{AAD}, \text{INFO}$}
09 \quad \textbf{for } (c, K') \leftarrow \text{AuthDecap}(sk_i^1, pk_i^1)
10 \quad \quad K' \leftarrow K_2
11 \quad \quad E' \leftarrow E' \cup \{(pk^j, pk^i, c', K')\}
12 \quad \quad (c_1, K) \leftarrow \text{AuthDecap}(sk_i^1, pk_i^1)
13 \quad \textbf{if } K \neq K_2
14 \quad \quad \textbf{return } E' \cup \{(pk^j, pk^i, c', K')\}
15 \quad \quad \textbf{else if } (c', (pk^j, pk^i, \text{AAD}, \text{INFO}) \in A
16 \quad \quad \quad \textbf{absorb}
17 \quad \quad \textbf{else}
18 \quad \quad \quad \textbf{return } E' \cup \{(pk^j, pk^i, (c', c''), \text{AAD}, \text{INFO})
19 \quad \quad \textbf{return } c', c''
```

\[ \]
D Proof for the AKEM Constructions

D.1 Proof of Theorem 7

Proof. We start with the Insider-CCA game for AKEM\textsuperscript{ExtH}[\text{KEM, SIG, H}] as Game G\textsubscript{0} in Listing 35:

\[ \Pr[(n, q_c, q_d, q_e, r_{sk})-\text{Insider-CCA}(A) \Rightarrow 1] = \Pr[G_0 \Rightarrow 1]. \]

Listing 35: Games G\textsubscript{0} – G\textsubscript{3} for the proof of Theorem 7.

\[
\begin{array}{ll}
\text{Game G}_1. & \text{In Game G}_1, \text{we change the challenge oracle such that query results are not only stored in set } \mathcal{E} \text{ in the case of } b = 1 \text{ but also for } b = 0. \text{ Due to the correctness of the decapsulation algorithm, this change does not effect the outputs of any query and the winning probabilities do not change. These changes are only conceptual resulting in } \\
& \Pr[G_0 \Rightarrow 1] = \Pr[G_1 \Rightarrow 1]. \\
\text{Game G}_2. & \text{In Game G}_2, \text{the KEM secret in the challenge oracle is replaced by uniformly random value from the KEM key space } K' \text{ and the sampled values are stored. The decapsulation oracle is also modified to provide consistent outputs. The advantage of this change corresponds to a CCA adversary against KEM depicted in Listing 36. Note that a KEM adversary can simulate the complete game including the RepSK oracle and its}
\end{array}
\]

\[
\begin{array}{ll}
& \text{Oracle CHALL}(i \in [n], j \in [n]) \\
& 25 \quad \text{if } j \in I_{pk} \\
& 26 \quad \text{return } 1 \\
& 27 \quad \{c, K' \} \leftarrow \text{Encaps}(pk_i) \\
& 28 \quad K' \leftarrow K \\
& 29 \quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_j, c, K')\} \\
& 30 \quad \sigma \leftarrow \text{Sign}(\text{sigk}, c|pk_i|pk_j|vk_j) \\
& 31 \quad K \leftarrow H(K', \sigma|pk_i|vk_j|vk_j) \\
& 32 \quad K' \leftarrow K \\
& 33 \quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, vk_j, \mathcal{E})\} \\
& 34 \quad \text{return } ((c, \sigma), K) \\
& \text{Oracle RepSK}(i \in [n], (sk, sigk)) \\
& 36 \quad (ak, \text{sigk}) \leftarrow (sk, \text{sigk}) \\
& 37 \quad (pk_i, vk_j) \leftarrow (\mu(sk), \mu(\text{sigk})) \\
& 38 \quad I_{sk} \leftarrow I_{sk} \cup \{i\}
\end{array}
\]
To prove the claim, we construct an adversary in Listing 37. Adversary $OChall$ and $\text{ODecap}$ different keys and at most one evaluation query for each decapsulation as well as challenge $K$. The game is modified such that the challenge oracle always returns ⊥ in case RepSK was called on the receiver’s key. In particular, we have

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \text{Adv}_{B,KEM}^{(n,q_k,q_r)\text{-CCA}}.$$ 

Listing 36: Adversary $B$ against CCA security of KEM having access to oracles $\text{ODecap}$ and $\text{OChall}$ using adversary $A$ from Games $G_1/G_2$ of the proof of Theorem 7.

```
Gr1
for i \in [n]
    (\text{sig}_i, \text{vk}_i) \leftarrow \text{Gen} 
\$E, \$F_p \leftarrow \emptyset$
\$b \leftarrow \{0, 1\}$
\$b' \leftarrow A_{\text{ADecap}, \text{ADecap}, \text{ODecap}, \text{RepSK}}((p_k_1, \text{vk}_1), \ldots, (p_k_n, \text{vk}_n))$
\$\text{Oracle } \text{ODecap}(i \in [n], (p_k, \text{vk}))$
    (c, K') \leftarrow \text{Encaps}(p_k_i)
    \sigma \leftarrow \text{Sign}(\text{sig}_i, c||p_k||\text{vk})
    K' \leftarrow H(K', \sigma||p_k||\text{vk}||c)
\$\text{return } [b = b']$\
\$\text{Oracle } A_{\text{Encaps}}(i \in [n], (p_k, \text{vk}))$
    (c, K') \leftarrow \text{Encaps}(p_k_i)
    \sigma \leftarrow \text{Sign}(\text{sig}_i, c||p_k||\text{vk})
    K' \leftarrow H(K', \sigma||p_k||\text{vk}||c)
\$\text{return } ((c, \sigma), K)$\
\$\text{Oracle } A_{\text{Decap}}((p_k, \text{vk}, j \in [n], (c, \sigma))$
    if $\exists K : ((p_k, \text{vk}), (p_k_j, \text{vk}_j), (c, \sigma), K) \in E'$
        return $K$
    if $\text{Vfy}(\text{vk}, c||p_k||\text{vk}_j, c, \sigma) \neq 1$
        return $\bot$
    if $b = 1$
        return $\bot$
    $K' \leftarrow \text{Decaps}(\text{vk}_j, c)$
    else
        $K' \leftarrow \text{ODecap}(j, c)$
        $K \leftarrow H(K', \sigma||p_k||\text{vk}||c, \text{vk}_j)$
    return $K$
```

Game $G_3$. In $G_3$, the game is modified such that the challenge oracle always returns a uniformly random key $K$ (Line 30), i.e. the output of $H$ is uniformly random. Additionally, the decapsulation oracle is changed in a similar way by replacing the output of $H$ by uniformly random value in case there is an element in set $E'$ (Line 16). The difference can be turned into an PRF adversary $C$ against $H$ such that:

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \text{Adv}_{C,H}^{q_k, q_{\text{sig}}\text{-PRF}}.$$ 

To proof the claim, we construct an adversary in Listing 37. Adversary $C$ can keep track of the key they queried by using index $\ell$ which is increased for every challenge query corresponding to a new key of the PRF game. This key can be reused if there is a decapsulation query which can be checked by modifying set $E'$ to not contain the actual key $K'$ but index $\ell$ of the key (Line 26 and Line 16). Adversary $C$ needs at most $q_{\text{sig}}$ many different keys and at most one evaluation query for each decapsulation as well as challenge query.

```
Gr2
for i \in [n]
    (\text{sig}_i, \text{vk}_i) \leftarrow \text{Gen} 
\$E, \$F_p \leftarrow \emptyset$
\$b, \ell \leftarrow \{0, 1\}$
\$b' \leftarrow A_{\text{ADecap}, \text{ADecap}, \text{ODecap}, \text{RepSK}}((p_k_1, \text{vk}_1), \ldots, (p_k_n, \text{vk}_n))$
\$\text{Oracle } \text{ODecap}(i \in [n], j \in [n])$
    if $j \in \ell_p$
        return $\bot$
    \$b, \ell \leftarrow \{0, 1\}$
    $K' \leftarrow \text{Encaps}(p_k_i)$
    \$\text{challenge } \text{distinct } \sigma \leftarrow \text{Sign}(\text{sig}_i, c||p_k||\text{vk})$
    $K' \leftarrow H(K', \sigma||p_k||\text{vk}||c, \text{vk}_j)$
    if $b = 1$
        return $\bot$
    $K' \leftarrow \text{Decaps}(\text{vk}_j, c)$
    else
        $K' \leftarrow \text{ODecap}(j, c)$
        $K \leftarrow H(K', \sigma||p_k||\text{vk}||c, \text{vk}_j)$
    return $K$
```

query: \( q_d + q_e \). Note that in the decapsulation oracle of \( \mathcal{G}_3 \), a new key \( K \) is chosen no matter if the same value was queried before in the challenge oracle, i.e., same key \( K' \) and same input \( \sigma || pk || vk || pk_j || vk_j \). However, if this was the case, this would lead to a return in Line 12 of the decapsulation oracle.

Game \( \mathcal{G}_3 \) is now independent of the challenge bit \( b \) since a uniformly random value is output in both cases:

\[
\Pr[\mathcal{G}_3 \Rightarrow 1] = \frac{1}{2}.
\]

The stated bound can be obtained by combining the game differences.

**Listing 37:** Adversary \( \mathcal{C} \) against PRF security of \( H \) having access to oracle \( \text{Eval} \) using adversary \( \mathcal{A} \) from Games \( \mathcal{G}_2/\mathcal{G}_3 \) of the proof of Theorem 7.

<table>
<thead>
<tr>
<th>C\text{EVAL}</th>
<th>Oracle \text{CHALL}(i \in [n], j \in [n])</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 for ( i \in [n] )</td>
<td>22 if ( j \in \mathcal{I}_k )</td>
</tr>
<tr>
<td>02 ((sk_i, \text{sigk}_i), (pk_j, \text{vk}_j)) &amp; \ Gen</td>
<td>23 return \bot</td>
</tr>
<tr>
<td>03 ( i \in [n] )</td>
<td>24 ((c, K') ) &amp; Encaps(pk_k) &amp; new key</td>
</tr>
<tr>
<td>04 ( i \in [n] )</td>
<td>25 ( i \leftarrow i + 1 )</td>
</tr>
<tr>
<td>05 ( b \leftarrow {0, 1} )</td>
<td>26 ( E' \leftarrow E \cup ({pk_j, c, i}) )</td>
</tr>
<tr>
<td>06 ( b \leftarrow A\text{DECAP,ADDECR,CHALL,REPSK}({pk_j, \text{vk}<em>j}, \ldots, {pk</em>{k}, \text{vk}_{k}}) )</td>
<td>27 ( \sigma \leftarrow \text{Sign(sigk_j, c</td>
</tr>
<tr>
<td>07 return [b = b']</td>
<td>28 ( K \leftarrow \text{Eval}(\ell, \sigma</td>
</tr>
<tr>
<td>Oracle \text{AENCAP}(i \in [n], {pk, \text{vk}})</td>
<td>29 if ( b = 1 )</td>
</tr>
<tr>
<td>08 ((c, K') ) &amp; Encaps(pk)</td>
<td>30 ( K \leftarrow K' )</td>
</tr>
<tr>
<td>09 ( \sigma \leftarrow \text{Sign(sigk_j, c</td>
<td></td>
</tr>
<tr>
<td>10 ( K \leftarrow H(K', \sigma</td>
<td></td>
</tr>
<tr>
<td>11 return ((c, \sigma), K')</td>
<td>Oracle \text{REPSK}(i \in [n], (sk, \text{sigk}))</td>
</tr>
<tr>
<td>Oracle \text{ADDECR}(pk, \text{vk}) &amp; {\in \mathcal{C}, j \in [n], (c, \sigma)}</td>
<td>33 ((sk_i, \text{sigk}_i) \leftarrow (sk_i, \text{sigk}) )</td>
</tr>
<tr>
<td>12 if ( \exists: (pk_j, c, \ell) \in E' )</td>
<td>34 ((pk_j, \text{vk}_j) \leftarrow (\mu(\text{sk}), \mu'(\text{sigk})) )</td>
</tr>
<tr>
<td>13 ( K \leftarrow \text{Decaps}(sk_i, c) )</td>
<td>35 ( \mathcal{I}_k \leftarrow \mathcal{I}_k \cup {i} )</td>
</tr>
<tr>
<td>14 return ( K )</td>
<td>□</td>
</tr>
</tbody>
</table>

D.2 Proof of Theorem 8

**Proof.** We start with the Outsider-Auth game for \( \text{AKEM} \) as Game \( \mathcal{G}_0 \) in Listing 38:

\[
\Pr[\mathcal{G}_0 \Rightarrow 1] = \Pr[(n, q_e, q_d)\text{-Outsider-Auth} \Rightarrow 1].
\]
The Pre-Shared Key Modes of HPKE

Listing 38: Games $G_0 - G_1$ for the proof of Theorem 8.

\begin{verbatim}
01 for i ∈ [n]
02 ((sk_i, sig_k_i), (pk_i, vk_i)) ← Gen
03 E ← ∅
04 b ← {0,1}
05 b’ ← AEncap\_Oracle\(A\)
06 return [b = b’]

Oracle AEncap\(i\) ∈ [n], (pk, vk)
07 (c, K)’ ← Encap\(pk\)
08 σ ← Sign(sig_k_i, c|pk||vk)
09 K ← H(K’, σ, pk_i)
10 E ← E ∪ \{((pk_i, sk_i), (pk, vk), (c, σ), K)\}
11 return ((c, σ), K)

Oracle A\_Decap\(j\) ∈ [n], (c, σ)
12 if ∃K : ((pk, vk), (pk_j, vk_j), (c, σ), K) ∈ E
13 then return K
14 if ∀b(σ_j|vk_j||sk_j, σ) ≠ 1
15 K ← ⊥
16 else
17 K’ ← Decaps\(sk\)
18 K ← H(K’, σ, pk)
19 if (pk, vk) ∈ {(pk_j, vk_j), . . . (pk_n, vk_n)} \⊥
20 abort
21 if b = 1 ∧ (pk, vk) ∈ {(pk_1, vk_1), . . . (pk_n, vk_n)} ∧ K ≠ ⊥
22 K’ ← K
23 E ← E ∪ \{((pk, vk), (pk_j, vk_j), (c, σ), K)\}
24 return K
\end{verbatim}

Game $G_1$. In $G_1$, the decapsulation oracle is modified such that the game aborts in case of a valid signature and honest sender keys. The difference can be turned into an SUF-CMA adversary $B$ against SIG:

$$\text{Pr}[G_0 ⇒ 1] − \text{Pr}[G_1 ⇒ 1] ≤ \text{Adv}^{(n, q_e)}_{B, \text{SIG}}$$

The adversary is depicted in Listing 39. Analyzing the winning condition of adversary $B$, we can observe that signature $σ$ is valid for the output message due to the check in Line 14. Moreover, the returned tuple $(i, c||pk_j||vk_j, σ)$ cannot be in the set of queried signature $Q$ (Game Listing 8): if it was in the set, there was a query to OSIG on inputs $i, c||pk_j||vk_j$ returning $σ$. This means that there is also an entry $((pk_i, vk_i), (pk_j, vk_j), (c, σ), K)$ in $E$ due to Line 10 leading to a return in Line 13. Hence, if adversary $B$ returns in Line 20, they win the SUF-CMA game. Thus, distinguishing the games can be turned into an adversary against SUF-CMA for $n$ users issuing at most $q_e$ signature queries.

Game $G_1$ is now independent of challenge bit $b$ since in the case of honest sender keys the game aborts and otherwise the condition in Line 21 is not true

$$\text{Pr}[G_1 ⇒ 1] = \frac{1}{2}.$$ 

Putting everything together yields the stated bound.

\[\Box\]

D.3 Proof of Theorem 9

Proof. We start with the IND-CCA experiment against AKEMs for the construction AKEM\(\text{NIKE}[\text{NIKE}, H]\) which we call Game $G_0$.

In Game $G_1$, we add the output of a challenge query to set $E$ in any case to provide consistent outputs later. This change does not influence the winning probability. Further, we replace the output of the second shared key invocation in the challenge oracle by a uniformly random output of the respective domain. We show that the difference of
Listing 39: Adversary $B$ against SUF-CMA with access to oracle OSign using adversary $A$ against $G_0/G_1$ from the proof of Theorem 8.

<table>
<thead>
<tr>
<th>$\mathcal{G}^{\text{OSign}}(\sigma_1, \ldots, \sigma_n)$</th>
<th>Oracle $\text{ADECAP}(pk, \sigma)$, $j \in [n]$, $(c, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 for $i \in [n]$</td>
<td>12 if $\exists K : ((pk, \sigma), (pk_j, \sigma_j), (c, \sigma), K) \notin \mathcal{E}$</td>
</tr>
<tr>
<td>02 $(sk_i, pk_i) \leftarrow \text{KEM.Gen}$</td>
<td>13 return $K$</td>
</tr>
<tr>
<td>03 $E, E' \leftarrow \emptyset$</td>
<td>14 if $\forall y_i, c_i</td>
</tr>
<tr>
<td>04 $b \leftarrow (0,1)$</td>
<td>15 $K \leftarrow 1$</td>
</tr>
<tr>
<td>05 $b' \leftarrow \mathcal{A}^{\text{ADECAP}, \text{ADECAP}}((pk_1, \sigma_1), \ldots, (pk_n, \sigma_n))$</td>
<td>16 else</td>
</tr>
<tr>
<td>06 return $\bot$</td>
<td>17 $K' \leftarrow \text{Decaps}(sk_j, c)$</td>
</tr>
<tr>
<td>Oracle $\text{AENCAP}(i \in [n], (pk, \sigma))$</td>
<td>18 $K \leftarrow H(K', \sigma, pk)$</td>
</tr>
<tr>
<td>07 $(c, K') \leftarrow \text{Encaps}(pk)$</td>
<td>19 if $\exists (pk, \sigma) = (pk_i, \sigma_i)$</td>
</tr>
<tr>
<td>08 $\sigma' \leftarrow \text{OSign}(i, c</td>
<td></td>
</tr>
<tr>
<td>$\text{\textit{signing oracle}}$</td>
<td>20 $\text{return } (i, c</td>
</tr>
<tr>
<td>09 $K \leftarrow H(K', \sigma, pk_i)$</td>
<td>/ return forgery</td>
</tr>
<tr>
<td>10 $E \leftarrow E \cup {((pk_i, \sigma_i), (pk, \sigma_i), (c, \sigma), K)}$</td>
<td>21 if $b = 1 \land (pk, \sigma) \in {(pk_i, \sigma_i), \ldots, (pk_n, \sigma_n)}$ and $K \neq \bot$</td>
</tr>
<tr>
<td>11 return $(c, \sigma, K)$</td>
<td>22 $K \leftarrow K'$</td>
</tr>
<tr>
<td></td>
<td>23 $E \leftarrow E \cup {((pk, \sigma), (pk, \sigma_j), (c, \sigma), K)}$</td>
</tr>
<tr>
<td></td>
<td>24 return $K$</td>
</tr>
</tbody>
</table>

the second step is indistinguishable assuming active security of NIKE. This is done by constructing an Active adversary against NIKE simulating game $G_0$ (in the adversary’s own real case) or $G_1$ (in the adversary’s own random game). We depict the construction of adversary $B$ against Active security of NIKE using adversary $A$ from game $G_0/G_1$ against Insider-CCA security of $\text{AKEMKNIKE}[\text{NIKE, H}]$ in Listing 41. This leads to

$$\left| \Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1] \right| \leq \text{Adv}_{B, \text{NIKE}}^{\text{Active}}.$$ 

In Game $G_2$, we replace the output of $H$ in the challenge oracle by a uniformly random output of the respective domain. This change can be upper bounded by the advantage of a PRF adversary against $H_2$ since the second input, $K_2$, is uniformly random. We obtain

$$\left| \Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1] \right| \leq \text{Adv}_{C, H_2}^{\text{PRF}}.$$ 

The resulting game has winning probability $\frac{1}{2}$ since $b$ is hidden from the adversary. Combining the game differences yields the stated bound. $\square$

D.4 Proof of Theorem 10

Proof. The proof uses a similar strategy as the proof for Theorem 9. We start with the Outsider-Auth game for $\text{AKEMKNIKE}[\text{NIKE, H}]$ in Listing 42 as $G_0$:

$$\left| \Pr[G_0 \Rightarrow 1] - \frac{1}{2} \right| = \text{Adv}_{A, \text{AKEMKNIKE}[\text{NIKE, H}]}^{\text{Outsider-Auth}}.$$ 

Game $G_1$. In Game $G_1$, we modify the encapsulation oracle by replacing the first NIKE shared key $K_1$ by a uniformly random value from the key space (Line 21) if the receiver key is honest (Line 20) and save the result together with the public keys in set $A$. In case there already exists a matching element in $A$, we use that one instead (Line 13). The same modifications are applied to the decapsulation oracle with the difference that the replacement with a uniformly random value is only done if the sender’s key $pk$ is honest.
and the original AKEM key $K$ is not $\perp$ (Line 41). The game difference is at most the advantage of an adversary $\mathcal{B}$ against Active security of NIKE:

$$|\Pr[\mathcal{G}_0 \Rightarrow 1] - \Pr[\mathcal{G}_1 \Rightarrow 1]| \leq \operatorname{Adv}_{\mathcal{B},\operatorname{NIKE}}^{(n,q_e+q_d,0,0,q_e+q_d,q_e+q_d)}.$$  

Adversary $\mathcal{B}$ is formally constructed in Listing 43. In the real case of the Active game of adversary $\mathcal{B}$, they perfectly simulate $\mathcal{G}_0$ since every call to the NIKE oracles returns the actual shared key $K_1$. In the random case, $\text{TEST}$ returns a uniformly random output when queried on $i, j$. The test oracle $\text{TEST}$ is only called for honest receiver (sender) keys in the encapsulation (decapsulation) oracle thus simulating Game $\mathcal{G}_1$ for adversary $\mathcal{A}$. Note that $\text{TEST}$ returns the same value when queried multiple times on the same indices $i, j$ which corresponds to the check for elements in set $\Lambda$ in Game $\mathcal{G}_1$.

**Game $\mathcal{G}_2$.** In Game $\mathcal{G}_2$, a set $\mathcal{H}$ is introduced which stores the output and all the inputs of queries to $\mathcal{H}$. Additionally, flag $\text{BAD}$ is set to $\text{true}$ and the game aborts if $\mathcal{H}$ is queried on the same inputs twice, i.e. there already exists a matching element in $\mathcal{H}$, in the case of a matching element in $\Lambda$. The game difference is

$$|\Pr[\mathcal{G}_1 \Rightarrow 1] - \Pr[\mathcal{G}_2 \Rightarrow 1]| \leq q_e(q_e+q_d)P_{\text{NIKE}}.$$  

The $\text{abort}$ case in the decapsulation never occurs. If there was a query on the same inputs to $\mathcal{H}$ in oracle $\text{ADECAP}$, this includes the last input $pk||pk_j||c$ and therefore there must also be a matching element in $\mathcal{E}$, the oracle would have returned in Line 29 and the
Listing 41: Adversary $B$ against Active security of NIKE using an adversary $A$ against $G_0/G_1$ for the proof of Theorem 9.

```
for $i \in [n]$ 01
    $sk_i \leftarrow \bot$ 02
    $pk_i \leftarrow \text{REGHonest}(i)$ 03
    $\ell \leftarrow n$ 04
    $\mathcal{E}, \mathcal{F}_n \leftarrow \emptyset$ 05
    $b, b^\ast \leftarrow \text{AEncap,ADecap,Chall,RepSK}(pk_1, \ldots, pk_n)$ 06
    return $b$ 07

$\text{AEncap}(i \in [n], pk)$ 08
if $sk_i = \bot$ 09
    if $\exists j : pk_j = pk$ 10
        $K_i \leftarrow \text{HonestRev}(i, j)$ 11
    else 12
        $\ell \leftarrow \ell + 1$ 13
        $\text{RegCorrupt}(\ell, pk)$ 14
        $K_i \leftarrow \text{CorruptRev}(i, \ell)$ 15
    else 16
        $K_i \leftarrow \text{NIKE.SharedKey}(sk_i, pk)$ 17
        $(sk^*, pk^*) \leftarrow \text{NIKE.KeyGen}$ 18
        $K_2 \leftarrow \text{NIKE.SharedKey}(sk^*, pk)$ 19
        $K \leftarrow H(K_1, K_2, pk || pk^*)$ 20
        return $(pk^*, K)$ 21

$\text{RepSK}(i \in [n], sk)$ 22
$(sk_i, pk_j) \leftarrow (sk, \mu(sk))$ 23
$I_{pk} \leftarrow I_{pk} \cup \{i\}$ 24

$\text{ADecap}(pk, j \in [n], c)$ 25
if $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$ 26
    return $K$ 27
if $sk_j = \bot$ 28
    if $\exists i : pk_i = pk$ 29
        $K_1 \leftarrow \text{HonestRev}(i, j)$ 30
    else 31
        $\ell \leftarrow \ell + 1$ 32
        $\text{RegCorrupt}(\ell, pk)$ 33
        $K_1 \leftarrow \text{CorruptRev}(\ell, j)$ 34
    else 35
        $K_1 \leftarrow \text{NIKE.SharedKey}(sk_j, pk)$ 36
        $K_2 \leftarrow \text{NIKE.SharedKey}(sk_j, c)$ 37
        $K \leftarrow H(K_1, K_2, pk || pk_j || c)$ 38
        return $K$ 39

$\text{Chall}(i \in [n], j \in [n])$ 40
if $j \in I_{pk}$ 41
    return $\bot$ 42
if $sk_i = \bot$ 43
    if $\exists i : pk_i = pk$ 44
        $K_1 \leftarrow \text{HonestRev}(i, j)$ 45
    else 46
        $K_1 \leftarrow \text{CorruptRev}(i, j)$ 47
    else 48
        $K_1 \leftarrow \text{NIKE.SharedKey}(sk_i, pk_j)$ 49
        $K_2 \leftarrow \text{Test}(\ell, j)$ 50
        $K \leftarrow H(K_1, K_2, pk || pk) || sk^*$ 51
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk, pk_j, pk^*, K)\}$ 52
return $(pk^*, K)$ 53
```
Listing 42: \( G_0 - G_3 \) for the proof of Theorem 10.

\[
\begin{align*}
\text{Oracle ADEcap}(pk \in PK', j \in [n], c) \\
\text{28} & \text{ if } \exists K : (pk, pk_j, c, K) \in \mathcal{E} \\
\text{29} & \text{ return } K \\
\text{30} & K_1 \leftarrow \text{NIKE.SharedKey}(sk_j, pk) \\
\text{31} & K_2 \leftarrow \text{NIKE.SharedKey}(sk_k, c) \\
\text{32} & K \leftarrow H(K_1, K_2, pk||pk) || c \\
\text{33} & \text{if } \exists K' : (K'_i, (pk, pk_j)) \in \mathcal{A} / \mathcal{G}_1 - G_3 \\
\text{34} & K_i \leftarrow K'_i \\
\text{35} & K \leftarrow H(K_1, K_2, pk||pk_j)|| c \\
\text{36} & \text{if } \exists b : (h, K_1, K_2, pk||pk_j)|| c \in \mathcal{H} / \mathcal{G}_2 - G_3 \\
\text{37} & \text{BAD } \leftarrow \text{true: abort} / \mathcal{G}_2 - G_3 \\
\text{38} & \text{else} / \mathcal{G}_2 - G_3 \\
\text{39} & K \leftarrow K' / \mathcal{G}_2 - G_3 \\
\text{40} & H \leftarrow H \cup \{(K, K_1, K_2, pk||pk_j)|| c\} / \mathcal{G}_2 - G_3 \\
\text{41} & \text{else if } pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \bot / \mathcal{G}_1 - G_3 \\
\text{42} & K_1 \leftarrow \text{KeyGen}(pk) / \mathcal{G}_1 - G_3 \\
\text{43} & A \leftarrow A \cup \{(K_1, pk_j)|| c\} / \mathcal{G}_1 - G_3 \\
\text{44} & K \leftarrow H(K_1, K_2, pk||pk_j)|| c / \mathcal{G}_1 - G_3 \\
\text{45} & K \leftarrow K' / \mathcal{G}_1 - G_3 \\
\text{46} & H \leftarrow H \cup \{(K, K_1, K_2, pk||pk_j)|| c\} / \mathcal{G}_1 - G_3 \\
\text{47} & \text{if } b = 1 \wedge pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \bot / \mathcal{G}_1 - G_3 \\
\text{48} & K \leftarrow K' / \mathcal{G}_1 - G_3 \\
\text{49} & E \leftarrow E \cup (pk, pk_j, c, K) / \mathcal{G}_1 - G_3 \\
\text{50} & E \leftarrow E \cup (pk, pk_j, c, K) / \mathcal{G}_1 - G_3 \\
\text{51} & \text{return } K / \mathcal{G}_1 - G_3 \\
\end{align*}
\]

Listing 43: Adversary \( B \) against Active security of NIKE using an adversary \( A \) against \( G_0/G_1 \) for the proof of Theorem 10.

\[
\begin{align*}
&\text{Oracle ADEcap}(pk \in PK', j \in [n], c) \\
&\text{18} & \text{ if } \exists K : (pk, pk_j, c, K) \in \mathcal{E} \\
&\text{19} & \text{ return } K \\
&\text{20} & K \leftarrow \text{AuthDecap}(pk, sk_j, pk) \\
&\text{21} & K_1 \leftarrow \text{NIKE.SharedKey}(sk_k, pk) \\
&\text{22} & \text{if } \exists : pk = pk / \text{honest sender} \\
&\text{23} & K_1 \leftarrow \text{Test}(i, j) \\
&\text{24} & \text{else} / \text{corrupted sender} \\
&\text{25} & \ell \leftarrow \ell + 1 \\
&\text{26} & \text{RecCorrupt}(\ell, pk) \\
&\text{27} & K_1 \leftarrow \text{CorruptRev}(j, \ell) \\
&\text{28} & K_2 \leftarrow \text{NIKE.SharedKey}(sk_k, c) \\
&\text{29} & K \leftarrow H(K_1, K_2, pk||pk_j)|| c / \mathcal{G}_1 - G_3 \\
&\text{30} & \text{if } b = 1 \wedge pk \in \{pk_1, \ldots, pk_n\} \wedge K \neq \bot \\
&\text{31} & K \leftarrow K' / \mathcal{G}_1 - G_3 \\
&\text{32} & E \leftarrow E \cup (pk, pk_j, c, K) / \mathcal{G}_1 - G_3 \\
&\text{33} & \text{return } K / \mathcal{G}_1 - G_3 \\
\end{align*}
\]
**Abort** is not reached. For the **Abort** in the encapsulation oracle to trigger, there must be an element in \( \mathcal{H} \) for which especially \( pk^* \) matches. Since \( pk^* \) is randomly generated by \textsc{NIKE.KeyGen}, there must be a collision in two public keys. There are at most \( q_e + q_d \) elements in set \( \mathcal{H} \). Hence, for a single query to \textsc{AEncap} the probability can be upper bounded by the entropy of the \textsc{NIKE} public key and in total we obtain

\[
\Pr[\text{BAD} = \text{true}] \leq (q_e + q_d)P_{\text{NIKE}}.
\]

**Game G\(_3\).** In \( \text{G}_3 \), the output of \( \mathcal{H} \) is replaced by a uniformly random value and the result is stored in \( \mathcal{H} \). The difference can be turned into an adversary \( \mathcal{C} \) against \textsc{PRF} security of \( H_1 \), i.e. \( \mathcal{H} \) keyed on the first component:

\[
|\Pr[\text{G}_2 \Rightarrow 1] − \Pr[\text{G}_3 \Rightarrow 1]| \leq \text{Adv}_{\mathcal{C},\mathcal{H}_1}^{(q_e + q_d - q_d)}\text{PRF}.
\]

We depict adversary \( \mathcal{C} \) in Listing 44. Note that adversary \( \mathcal{C} \) needs potentially a new \textsc{PRF} key for each call to \textsc{AEncap} and \textsc{ADecap} and the same amount of queries to the \textsc{PRF} oracle \textsc{Eval}.

**Listing 44:** Adversary \( \mathcal{C} \) against \textsc{PRF} security of \( H_1 \) using an adversary \( \mathcal{A} \) against \( G_2/G_3 \) for the proof of Theorem 10.

\[
\begin{array}{ll}
\text{Listing 44:} & \text{Adversary } \mathcal{C} \text{ against PRF security of } H_1 \text{ using an adversary } \mathcal{A} \text{ against } G_2/G_3 \text{ for the proof of Theorem 10.}
\end{array}
\]

In the resulting game, challenge bit \( b \) is completely hidden and the winning probability is \( \frac{1}{2} \) from which the stated bound follows.